Solving Large Sparse Systems of Nonlinear Equations for the Simulation of Steam Cracking

Thesis · October 1998

DOI: 10.13140/2.1.2116.8324

1 author:

Nils van Velzen
Delft University of Technology
24 PUBLICATIONS **733** CITATIONS

SEE PROFILE

Solving Large Sparse Systems of Nonlinear Equations for the Simulation of Steam Cracking

Zoetermeer, October 1998

C. van Velzen
Hoge Rijndijk 94
2313KL Leiden
Tel. 071-215131783

Leiden University (RUL)
Department of Mathematics
Sub-Faculty Numerical Analysis

Kinetic Technology International (KTI) B.V.
Division Pyrotec
Bredewater 26
2715CA Zoetermeer
The Netherlands

Supervisors:
Dr. J. A. van de Griend (RUL)
Dr. ir. C. van Leeuwen (KTI)

PREFACE

This thesis reports about my research that was performed at KTI in Zoetermeer the Netherlands from January 1998 to July 1998. The aim of the research was to determine the feasibility of the OPEN SPYRO project.
I thank my parents for making it possible for me to study mathematics at Leiden University. I really enjoyed the collaboration with Marco van Goethem a student from the Technical University Delft who worked on the same project.
I want to thank Dr. ir. C. van Leeuwen for the supervision at KTI and Dr. J.A. van de Griend who supervised this project at Leiden University. Special thanks to Dr. ir. P. J. T.
Verheijen from the Technical University Delft for his remarks and advises.
Further I am grateful to all my colleagues at the Pyrotec division of KTI, in particular Barbie Hunt, Florian Kleinendorst and Peter Valkenburg.
I also appreciate the support of my girlfriend during this research.

He who involves himself too much with the insignificant
is incapable of doing greater things.

La Rochefoucauld

SUMMARY

Steam pyrolysis of ethane and naphtha is an important chemical bulk process. It produces ethylene and propylene, which are important base chemicals. In order to be competitive, crackers have to be operated at near optimal conditions. Hence, a simulation program of the process, particularly of the pyrolysis is very helpful. KTI uses and licenses such a program called SPYRO (TM). Development of this program has started over 20 years ago.
Consequently, it uses a closed model.

It has been the objective of this study to investigate the feasibility of the development of a solver for an open version of SPYRO. Here open means that the equations are written in residual form. This enhances the flexibility of the program very much.

The SPYRO model consists of a set of differential equations that describe the material-, heat- and momentum balances along the tubular reactor. The numerical solution of the ODE's is computed by using orthogonal collocation on finite elements. This leads to a system of nonlinear equations. The constructed model is solved with an equation solver.

Generally a nonlinear solver consists of an iterative procedure which yields a linear approximation based on the Jacobian of the model. Such procedures are Newton's, Broyden's and Schubert's method. They all use linear solvers. We have investigated several linear solvers: The LU-decomposition, Householder's method, the ILU GMRES and an in-house (KTI) solver. In order to get reasonable starting values, continuation methods have been analyzed, in particular Euler integration and Newton homotopy.

To test the solvers we have used the model of Froment for ethane cracking because the documentation to make an open SPYRO model was insufficient. This model was extended and made sufficiently stiff to imitate the SPYRO model.

It appears that the LU-decomposition performs best in terms of accuracy and computation time to solve the linear system. Moreover, this method performs best in combination with Broyden's method to solve the nonlinear system. The speed of solution of the linearized set of model equations depends on the size and the sparsity structure of the Jacobian. The latter has an enormous effect on the fill-in of the LU-decomposition.

If damping methods are used, the nonlinear solvers converge even with a bad initial guess. (Hence the pre-solvers were not really necessary.) The damping method due to Deuflhard results in the fastest convergence. For solving very stiff systems, we have to use domain damping with reinitialization to converge to the solution.

Scaling did not influence the performance of Newton's method. Scaling in the solution space significantly reduced the number of iterations for Broyden's method. For Schubert's method the opposite is true.

On the basis of the above results we may conclude that it is possible to develop a solver for the Open SPYRO model, which is accurate and robust. The computation time has to be improved. Ways to achieve this are by generating better starting values, redesign the source and optimizing the code.

TABLE OF CONTENTS

1 INTRODUCTION

The chemical process of steam pyrolysis for the production of ethylene is modeled for simulation purposes. Pyrolysis, cracking of hydrocarbons diluted with steam, takes place between 650 and 850°C. The reactor (see Figure 1) used for this process consists of two sections: a convection section, in which the hydrocarbon feed is mixed with steam and heated to the temperature where cracking starts (650°C), and the firebox section where the hydrocarbon feed flows through the radiant coils in which the cracking takes place. The residence time in the tubes is very short (0.5-1.0 s).

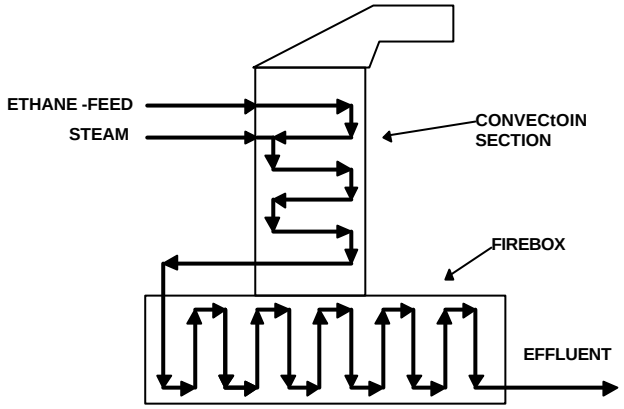


Figure 1: A schematically representation of an ethane cracker. In the convection section the feed is heated to a temperature of approximately 650°C. The cracking takes place in the fire box at a temperature between 650 - 850°C.

The production of ethylene is a large scale process with small profit margins. The plant must operate close to the optimal conditions to be competitive with other companies. It is not surprising that a lot of effort is put into the development of a simulation program. SPYRO® is a product marketed by PYROTEC, a division of KTI, to simulate the chemical processes within the radiant coil. The current model consists of a number of differential equations and algebraic equations. SPYRO® computes the simulation by directly integrating these equations. In the successor of SPYRO®, called Open SPYRO , PYROTEC wants to build in the possibility for optimization. To adapt the problem for optimization they want to use an open model. Therefore the cracking of the feed is now modeled by a system of sparse nonlinear equations.
The aim of this research is to build and test a solver for sparse systems of nonlinear equations. This solver is used to solve the nonlinear equations evolving from the simulation of steam cracking to produce ethylene. The constructed solver will be used for the development of Open SPYRO and must be robust and fast.
The solver is tested on models based on the ethane cracker model by FROMENT(1979). A number of variations of this model are tested. These models are described in Appendix C.

In Chapter 2 some information about the sparse system of nonlinear equations is given. The modeling of the processes within the radiant coil is not a part of this research. Therefore we will only present some aspects of the system of nonlinear equations which are important for the development of the solver for sparse systems of nonlinear equations.

The handling and solving of large sparse systems of linear equations differs from the methods used for small non-sparse systems of linear equations. By handling these sparse systems in a different way we gain computation time and memory. These aspects will be discussed in Chapter 3. In this chapter there are three methods presented to solve sparse systems of linear equations. These methods are: Gaussian elimination, Householder's method and an iterative method called the ILU-GMRES.

The aspects of solving the sparse systems of nonlinear equations are presented in Chapter 4. In this chapter we will discuss Newton's method and two methods based on Newton's method: Broyden's method and Schubert's method. For problems which are very hard to solve we can use Continuation methods to find proper initial approximation for Newton's Broyden's or Schubert's method to converge. Other aspects like scaling and damping are discussed too.

The structure of the created solver for sparse systems of nonlinear equations and the aspects of scaled norms and stopping criteria are discussed in Chapter 5.

The solver as described in Chapter 5 is tested on various models. The results of these tests are presented in Chapter 6.

The conclusion and the recommendations for further development can be found in Chapter 7.

2 THE MATHEMATICAL MODEL

2.1 Introduction

The modeling of the chemical reactions within the radiant coil is not a part of this research. In this chapter a short description of the mathematical model is given. The model describes the cracking inside the radiant coils of the hydrocarbon feed represented by:

- Mass balance (Component balances)
- Heat balance
- Momentum balance

As a result the process taking place in the radiant coil is described by a system of Differential Algebraic Equations (DAE).
The solution of this system of Differential Algebraic Equations is approximated by Orthogonal Collocation on Finite Elements (OCFE) (VILLADSEN(1978)). With OCFE we are able to convert the DAE into a set of sparse nonlinear algebraic equations which have to be solved.

2.2 The Structure of the Jacobian Matrix

The solution of the system of differential algebraic equations is approximated by polynomials. To get a better approximation, one could choose a higher order of polynomial to approximate the solution. Unfortunately the approximation can become worse if the order of polynomial increases. Therefore the order of the polynomial is not chosen too high but the tube is split into a number of sections in which the DAE is approximated. A method based on this principle is the method of Orthogonal Collocation of Finite Elements (OCFE). This method will be used to approximate the solution of the OFCE. To find this approximation to the OCFE we have to solve a system of nonlinear equations. The structure of the Jacobian matrix is schematically presented in Figure 2 and Figure 3.

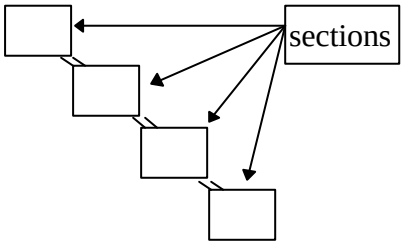


Figure 2: The schematically structure of the Jacobian matrix of the nonlinear system. In this example four sections are used to approximate the solution.

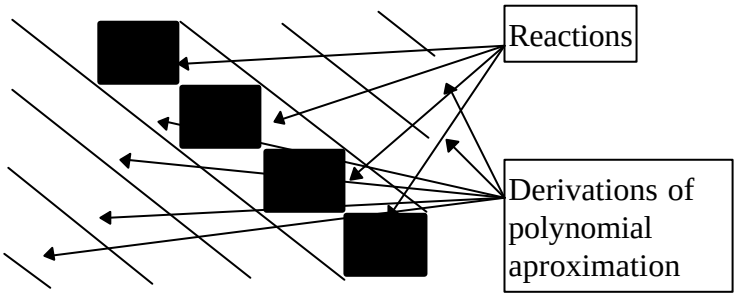


Figure 3: Structure of one section of the Jacobian matrix of the nonlinear system. In this example a polynomial of at most the fourth degree is used to approximate the solution in each section.

The number of blocks with reactions equals the degree of the polynomial approximation within the section. The example given in Figure 2 and Figure 3 has 4 sections and in each section the solution is approximated by a polynomial of at most degree 4.
Some models have a slightly different structure. In these models variables are defined which do not only influence one section. The structure of these Jacobian matrices are schematically given in Figure 4.

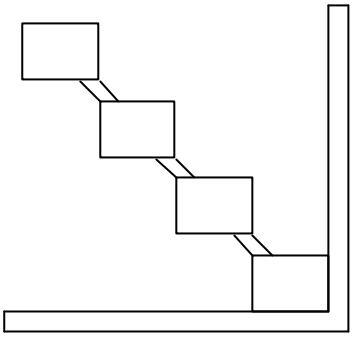


Figure 4: Generalized structure of the Jacobian matrix for the simulation of the steam cracking using orthogonal collocation on finite elements.

Depending on the number of sections and collocation points, the percentage of the non-zero elements in the Jacobian matrix varies between 0.001% and 10%. This system of equations can have a dimension up to 15,000. By storing and handling these matrices in a different way than we do with ordinary full-matrices we gain computation time and memory use.

2.3 Aspects of the Model

Chemical models are in general ill-posed and are hard to solve. The reason is the differences in magnitude between the different variables within the system.

The values of the different components in the solution of the cracker model vary between 10^{-6} and 10^{6}; therefore the system is very sensitive to computational errors. An other aspect of the equations is that there are a number of radical reactions within the SPYRO® model. The speed of these reactions is very high and leads to stiff differential equations, which are hard to solve using ordinary techniques for solving differential equations.

3 SPARSE SYSTEMS OF LINEAR EQUATIONS

3.1 Introduction

Most iterative methods for solving systems of nonlinear equations, solve a system of linear equations for every iteration. Therefore we need to have a good working linear solver before we can start building a solver for nonlinear equations.
A system of n linear equations can be written in the form:

$$\mathbf{Ax} = \mathbf{b}, \tag{3.1}$$

where $\mathbf{A}$ is a real $n \times n$ matrix, $\mathbf{b}$ a given n-dimensional vector and $\mathbf{x}$ the unknown n-dimensional vector.
There are several methods for finding the solution of (3.1). If $\mathbf{A}$ is not sparse, (3.1) is usually solved by using direct methods. Direct methods are in general robust and the computation costs are predictable. The drawback of direct methods is that if $\mathbf{A}$ is sparse it can produce huge amounts of fill-in. This is discussed further below.
The iterative methods form another class of methods. In general these methods need less memory than direct methods and can be very fast. The drawbacks of iterative methods are that we need an initial guess, we have to set different parameters for different linear systems, and convergence is not guaranteed.
Prior to discussing methods for solving systems of linear equations we will discuss some storage schemes for sparse matrices.

3.2 Storage of Sparse Matrices

If we store sparse matrices in the usual manner we need to store n^2 real numbers, for example if $n=10^4$ and the coefficients are stored in a double precision, we need 800 MB to store this matrix. Less storage is needed if the matrix is stored in a different way.
The different storage schemes are illustrated with the matrix

$$\mathbf{A} = \begin{pmatrix} 11 & 0 & 0 & 14 & 0 \\ 0 & 22 & 21 & 0 & 25 \\ 0 & 23 & 0 & 32 & 0 \\ 30 & 0 & 34 & 44 & 43 \\ 0 & 25 & 0 & 0 & 55 \end{pmatrix}. \tag{3.2}$$

This matrix is a non-symmetric with 13 nonzero elements. We will denote the number of non-zeros by the function

$$\text{non-zero}(\mathbf{A}). \tag{3.3}$$

3.2.1 Coordinate (COO) Format

The most simple way to store a sparse matrix $\mathbf{A}$ is by three one dimensional arrays (SAAD(1994)), all of the size non-zero($\mathbf{A}$):

-a real array, `value` containing the values of the non-zero elements of **A,**
-an integer array, `col` containing their column indices and,
-an integer array, `row` containing their row indices.

The order of the non-zero elements within the array is not important.

Example (COO Format) In the COO format the matrix (3.2) can be stored as:

`value`	55	23	21	11	25	34	25	43	22	30	44	32	14
`col`	5	2	3	1	2	3	5	5	2	1	4	4	4
`row`	5	3	2	1	5	4	2	4	2	4	4	3	1

3.2.2 Sorted Coordinate (SCOO) Format

The only difference between the SCOO and COO format is that the elements are stored in a sorted order.
We can store by row and by column. Storing by row means that the first row is stored first than the second etc.(within each row the elements are sorted by column). Storing by column goes in a similar way.
The advantage of this method is that elements are easier to locate than in the COO format.

Example (SCOO Format) Using the SCOO format (stored by row) the matrix (3.2) is stored as:

`value`	11	14	22	21	25	23	32	30	34	44	43	25	55
`col`	1	4	2	3	5	2	4	1	3	4	5	2	5
`row`	1	1	2	2	2	3	3	4	4	4	4	5	5

3.2.3 Compressed Row Storage (CRS) Format

A more efficient storage is the compressed row format. The Compressed row format is similar to the SCOO format stored by row (SAAD(1994)). There is only one difference, instead of the array `row` the n+1 dimensional array `row_index` is used. The i^{th} element of `row_index` denotes the number of nonzero elements+1 in the 1^{st} to $(i\text{-}1)^{th}$ row. By definition the first element of `row_index` equals 1 and the last equals non-zero(**A**)+1

Example (CRS Format) Using the CRS format (stored by row) the matrix (3.2) is stored as:

`value`	11	14	22	21	25	23	32	30	34	44	43	25	55
`col`	1	4	2	3	5	2	4	1	3	4	5	2	5

`row_index`	1	3	6	8	12	14

3.3 Direct Methods

Methods for solving systems of linear equations which are based on the factorization of the coefficient matrix are called direct methods. If no rounding errors are made, these methods produce the exact solution of a linear system of equations after a finite number of computations. Two well known direct methods are presented: Gaussian elimination and Householder's method.

3.3.1 Gaussian Elimination

The idea behind Gaussian elimination is that the matrix **A** is reduced to an upper triangular system, which can be solved directly by using a back-substitution. By adding multiples of rows to each other we systematically eliminate all elements below the diagonal of the matrix. In the first step of the elimination process, all elements below the diagonal of the first row are eliminated. In the 2nd step all elements below the diagonal of the 2nd row are eliminated etc. In n-1 steps we have decomposed the original matrix **A** into an upper triangular matrix. The pseudo code of the standard Gaussian elimination is given in Figure 5.

```
For k=1 to n-1
  For i=k+1 to n
    a_ik=a_ik/a_kk
    For j=k to n
      a_ij=a_ij-a_ik*a_kj
    Endfor
  Endfor
Endfor
```

Figure 5: The algorithm of standard Gaussian elimination.

The lower triangular matrix consists of the coefficients $l_{ij}=-a_{ik}/a_{kk}$ computed during the Gaussian elimination and ones on the diagonal. After this decomposition the solution is obtained by solving two triangular systems, **Ly**=**b** and **Ux**=**y**. A thorough description of this method can be found in DUFF(1986).

An important issue is the choice of the row that is used to eliminate a certain unknown. If during the elimination process the current row has a zero or very small entry on the diagonal, the algorithm breaks down or becomes numerically unstable. This can be avoided by interchanging rows or columns (partial pivoting) or interchanging rows and columns (complete pivoting) to get a 'better' element on the diagonal. There are matrices for which partial pivoting is unstable, but in practice it is considered to be stable. This reordering can also be done beforehand. However this priori reordering is usually performed to get a better suited sparsity structure rather then to ensure numerically stability(DUIN(1998)).

3.3.2 Householder Transformation

If we apply Gaussian elimination we decompose the matrix **A** into the product of two triangular matrices which are easy to invert. Another type of matrices which are easy to invert are the orthogonal matrices (**Q** is orthogonal then $\mathbf{Q}^{-1}=\mathbf{Q}^{T}$). The Householder transformation algorithm decomposes the matrix **A** into the product of an orthogonal matrix **Q** and an upper triangular matrix **R**. The matrix **Q** is in general not sparse. Therefore **Q** is not constructed explicitly for large sparse systems. The first Householder transformation is given by

(3.4) $$\mathbf{P}_1=\mathbf{I}-\frac{2}{\mathbf{v}_1^T\mathbf{v}_1}\mathbf{v}_1\mathbf{v}_1^T$$

with $\mathbf{v}_1 = a_1 + \frac{|a_{1,1}|}{a_{1,1}}|a_1|_2 e_1$, a_1 denotes the first column of **A** and $a_{1,1}$ the first element of this column .
If we compute $\mathbf{P}_1\mathbf{A}$, then the application of the Householder transformation results in a first column of zeros except for the first element. This procedure can be continued for the $(n-1)\times(n-1)$ sub-matrix and after *n*-1 steps the process is completed.
The condition number of a matrix does not change if it is multiplied by an orthogonal matrix, hence $\mathrm{cond}(\mathbf{A}) = \mathrm{cond}(\mathbf{R})$. Householder's method is numerically more stable than the Gaussian elimination because the condition number does not change during the decomposition algorithm.
For the dense case, the Householder transformation algorithm takes twice as much computations than Gaussian elimination.

3.4 Iterative Methods

Iterative methods for solving linear systems of equations start with some initial guess. From this initial guess a new approximation of the solution is computed. This procedure is repeated until convergence is reached. The most basic iterative method is the Richardson iteration:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}. \tag{3.5}$$

Unfortunately, this method only converges when the spectral radius (r) of **I-A** is less than one.
We select a matrix **M**, called the splitting matrix such that the preconditioned system

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{3.6}$$

is better conditioned than the original system. We have to choose **M** such that it is 'easy' to invert. This means that $\mathbf{M}\mathbf{x} = \mathbf{b}$ can be solved with less effort than the original system $\mathbf{A}\mathbf{x} = \mathbf{b}$. To reduce the number of iterations we have to find **M** close to **A** in the sense that the spectral radius of $\mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$ is smaller than the spectral radius of **I-A** . In the above we have considered left preconditioning. A different way of preconditioning is called right preconditioning where the preconditioned system is defined by

$$(\mathbf{A}\mathbf{M}^{-1})(\mathbf{M}\mathbf{x}) = \mathbf{b} \tag{3.7}$$

In the case of right preconditioning the system $(\mathbf{A}\mathbf{M}^{-1})\mathbf{y} = \mathbf{b}$ is solved by using an iterative method, and **x** is computed from the system $\mathbf{M}\mathbf{x} = \mathbf{y}$.

Some examples of left preconditioners for the Richerdson iteration are:

- **M=LU**: Because of computational errors the LU-decomposition is not exact. This method is called iterative refinement.
- $\mathbf{M}=\mathbf{D}_A$: The diagonal of the matrix **A** is chosen as preconditioner. This method is known as Gauss-Jacobi iteration.
- $\mathbf{M}=\mathbf{L}_A + \mathbf{D}_A$: with $\mathbf{L}_A$ the lower triangular part of **A**. This method is known as Gauss-Seidel iteration.

Unfortunately these choices do not always lead to a spectral radius less than one, therefore these methods fail to converge in general.

3.4.1 GMRES

A class of methods with better convergence properties are the iterative methods based on finding optimal approximations in the Krylov space (van der VORST(1997), GOLUB(1997)). We will discuss the GMRES (Generalized Minimum Residual Method). This method can be used for general sparse matrices. First we will construct the Krylov projection space.
The Richardson iteration can be written as

$$\mathbf{x}^{(i+1)} = \mathbf{b} + (\mathbf{I}-\mathbf{A})\mathbf{x}^{(i)} = \mathbf{x}^{(i)} + \mathbf{r}^{(i)}, \qquad \mathbf{r}^{(i)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(i)}. \tag{3.8}$$

Multiplication by -**A** and adding **b** gives

$$\mathbf{b} - \mathbf{A}\mathbf{x}^{(i+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(i)} - \mathbf{A}\mathbf{r}^{(i)}, \tag{3.9}$$

this is equivalent to

$$\mathbf{r}^{(i+1)} = (\mathbf{I}-\mathbf{A})\mathbf{r}^{(i)} = (\mathbf{I}-\mathbf{A})^{i+1}\mathbf{r}^{(0)} = \mathrm{P}_{i+1}(\mathbf{A})\mathbf{r}^{(0)}. \tag{3.10}$$

This means that for the error we have

$$\mathbf{A}(\mathbf{x}-\mathbf{x}^{(i+1)}) = \mathrm{P}_{i+1}(\mathbf{A})\mathbf{A}(\mathbf{x}-\mathbf{x}^{(0)}) \Rightarrow \mathbf{x}-\mathbf{x}^{(i+1)} = \mathrm{P}_{i+1}(\mathbf{A})(\mathbf{x}-\mathbf{x}^{(0)}). \tag{3.11}$$

If we choose $\mathbf{x}^{(0)}$=0, we get for the Richardson iteration

$$\mathbf{x}^{(i)} = \mathbf{r}^{(0)} + \mathbf{r}^{(1)} + \ldots + \mathbf{r}^{(i-1)} = \sum_{j=0}^{i-1} (\mathbf{I}-\mathbf{A})^j \mathbf{r}^{(0)}. \tag{3.12}$$

We define the Krylov subspace generated by **A** and $\mathbf{x}^{(0)}$

$$K^i(\mathbf{A};\mathbf{r}^{(0)}) = \{\mathbf{r}^{(0)}, \mathbf{A}\mathbf{r}^{(0)}, \ldots, \mathbf{A}^{i-1}\mathbf{r}^{(0)}\}. \tag{3.13}$$

Note: for $\mathbf{x}^{(0)}$ we can use every value. In this case the constructed Krylov subspace is slightly different than (3.13).

After every Richardson iteration we generate a Krylov subspace of increasing dimension. The question is whether we can select a better $\mathbf{x}^{(i)}$ from this subspace K^i. We would like to find a $\mathbf{x}^{(i)}$ for which $\left|\mathbf{x}^{(i)}-\mathbf{x}^*\right|_2$ is minimal, but this is in most cases not possible because $\mathbf{x}^*$ is not known.

An alternative is to select $\mathbf{x}^{(i)}$ for which $\left|\mathbf{b}-\mathbf{A}\mathbf{x}^{(i)}\right|_2$ is minimal. This method is called GMRES (Generalized Minimum Residual Method). For finding this optimal solution we have to solve a system of linear equations of size *i*. Therefore the GMRES algorithm is restarted after a number of steps. The last computed approximation of the preceding session of the GMRES algorithm is used as an initial value for the new session .

3.5 Scaling of Linear Systems of Equations

The condition number of $\mathbf{A}$, $\mathrm{cond}(\mathbf{A})$ plays an important role for the precision of the computation of the solution of $\mathbf{Ax}=\mathbf{b}$. Instead of solving $\mathbf{Ax}=\mathbf{b}$ it is possible to compute the solution of $\mathbf{D}_1\mathbf{A}\mathbf{D}_2\mathbf{y}=\mathbf{D}_1\mathbf{b}, \mathbf{x}=\mathbf{D}_2^{-1}\mathbf{y}$ for two 'easily' invertable matrices $\mathbf{D}_1$ and $\mathbf{D}_2$. By easily invertable we mean that systems of linear equations with these matrices can be computed fast. The matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ are chosen such that $\mathrm{cond}(\mathbf{D_1}\mathbf{A}\mathbf{D_2}) < \mathrm{cond}(\mathbf{A})$. There are methods available, PALOSCHI(1988), for computing optimal values for $\mathbf{D}_1$ and $\mathbf{D}_2$. Unfortunately these methods require the eigenvalues of **A** which are too costly to compute. In BAUER(1963) it is proved $\min_{D_1}\{\mathrm{cond}_\infty(D_1A)\}$ is given by:

(3.14)
$$D_1^{-1}=\begin{pmatrix} \sum_{j=1}^{n}|a_{1,j}| & 0 & \cdots & 0 \\ 0 & \sum_{j=1}^{n}|a_{1,2}| & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sum_{j=1}^{n}|a_{1,n}| \end{pmatrix}$$

and $\min_{D_2}\{\mathrm{cond}_\infty(AD_2)\}$ by

(3.15)
$$A^{-1}=\begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ b_{n,1} & \cdots & \cdots & b_{n,n} \end{pmatrix}, \quad D_2=\begin{pmatrix} \sum_{j=1}^{n}|b_{1,j}| & 0 & \cdots & 0 \\ 0 & \sum_{j=1}^{n}|b_{1,2}| & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sum_{j=1}^{n}|b_{1,n}| \end{pmatrix}$$

The inverse of the matrix is not known before applying Gaussian elimination, therefore scaling by $\mathbf{D}_2$ cannot be used for an arbitrary matrix. Scaling by the matrix $\mathbf{D}_2$ can be useful for iterative methods for solving nonlinear systems of equations, if we use the inverse of the linear system of the preceding iteration. Unfortunately this method cannot be applied for sparse systems because the inverse matrix cannot be computed explicitly.

3.6 Reordering the System of Linear Equations to Avoid Fill-in

The memory needed to solve large systems of linear equations by using direct methods can be huge because of the produced amount of fill-in. By simple row and column interchanges it is possible to reduce this fill-in dramatically. This is illustrated by the following system:
in Figure 6 the LU-decomposition of an upper arrow shaped matrix is shown for standard Gaussian elimination without pivoting. The number of elements of the computed LU-decomposition equals $n(n+1)$.

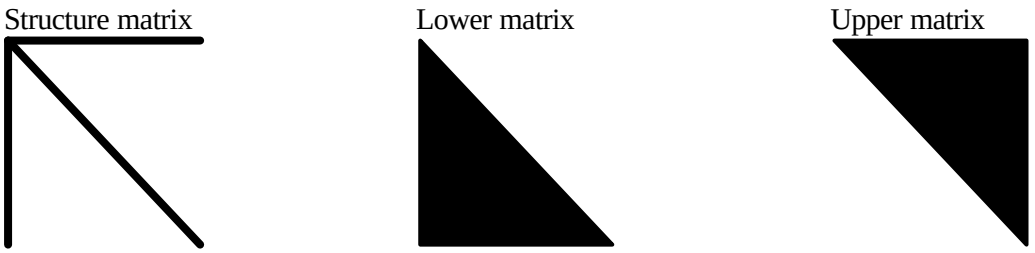


Figure 6: The LU-decomposition of upper 'arrow' matrix. In this figure it is shown that the LU-decomposition need the storage of a full matrix.

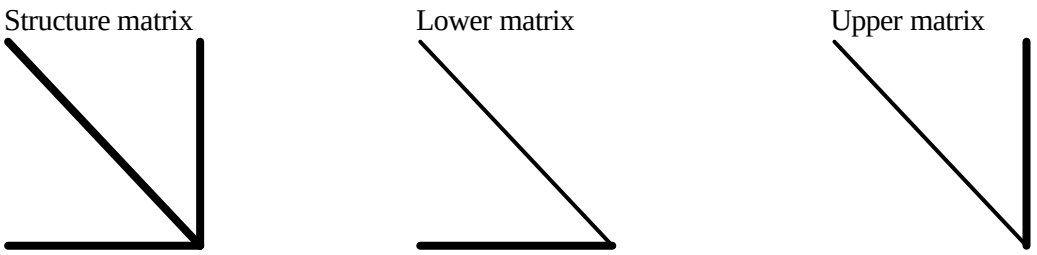


Figure 7: LU-decomposition of lower 'arrow' matrix. In this figure it is shown that the sparsity structure of the original matrix is preserved in its LU-decomposition.

If a few permutations are made we can construct from the same system of linear equations a lower arrow shaped matrix. The LU-decomposition of this matrix has the same structure as the lower shaped matrix and the storage needed for the LU-decomposition equals the storage needed for the original matrix as illustrated in Figure 7.

3.6.1 Desired Forms for Gaussian Elimination

Gaussian elimination behaves very good on diagonal matrices.
If Gaussian elimination without pivoting is used to decompose a (p,q) band-matrix (APPENDIX A) then L+U is a (p,q) band-matrix. If Gaussian elimination with partial pivoting is used to decompose a (p,q) band-matrix then L+U is a $(p+q,q)$ band-matrix or a $(p,p+q)$ band-matrix depending on the type of pivoting.

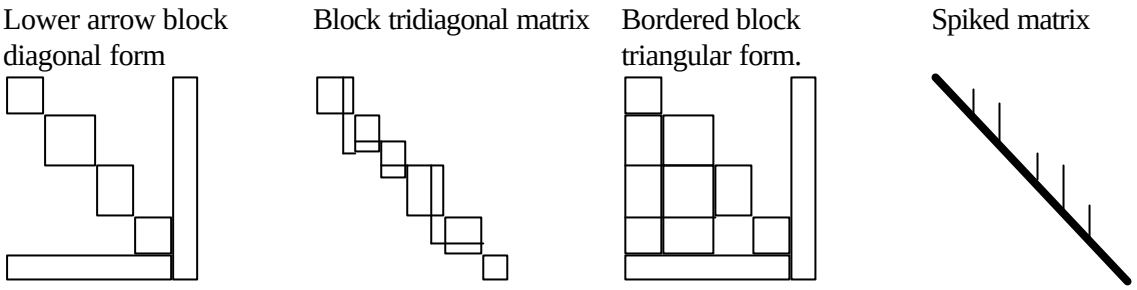


Figure 8: Matrixes which produce no fill-in for Gaussian elimination without pivoting.

In DUFF(1986) methods are described which minimize fill-in, by minimizing the bandwidth of the matrix. The ethylene cracker model is of the lower arrow block diagonal form and the maximum amount of fill-in can be computed beforehand.
Minimizing the fill-in within the blocks can be very useful. The sparsity structure of the Jacobian of ethylene cracker model PR3 produced huge amounts of fill-in within the blocks (see Figure 9). After rearranging the structure within the blocks the fill-in was reduced dramatically and the system was solved over ten times faster. The rearranged matrix and its LU-decomposition are presented in Figure 10.

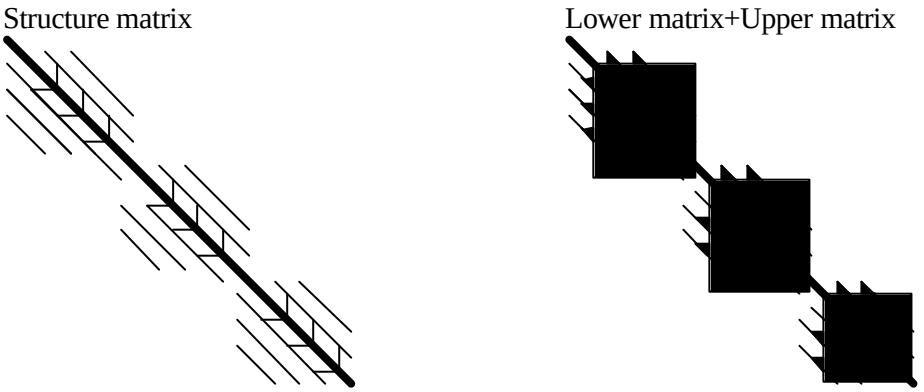


Figure 9: structure Jacobian and its LU-decomposition of the original PR3 model (schematically).

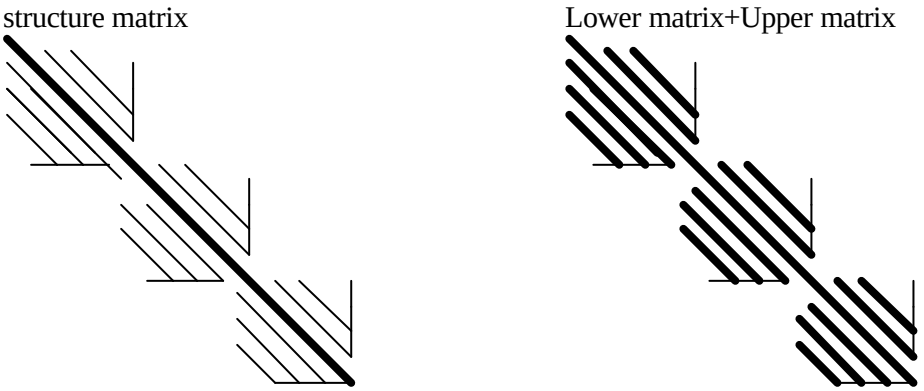


Figure 10: Structure Jacobian and its LU-decomposition of the adjusted PR3 model (schematically).

In TEWARSON(1973) a method for finding an optimal pivot element at every step of the Gaussian elimination to minimize the amount of fill-in produced in that particular step (local fill-in) is given. The drawback of this method is the fact that we have to multiply a sparse and a dense matrix. This product

is in general not sparse and can't be stored. The reordering options discussed in GILBERT(1991) are tested for our type of matrices. But no significant improvement was found.

3.6.2 Desired Form for Householder's Method

The Householder transformation is an outer-product of the first vector of the (sub)matrix. This vector must be as sparse as possible, therefore we must place vertical bands of non-zeros below the diagonal at the right hand side. If Householder's method is applied on a (p,q) band-matricix then **R** is an (0,p+q) band-matrix. For large systems of linear equations Householder's method is only useful for almost upper diagonal matrices. For arbitrary matrices huge amounts of fill in will occur.

Structure of original matrix Upper matrix by householder's method

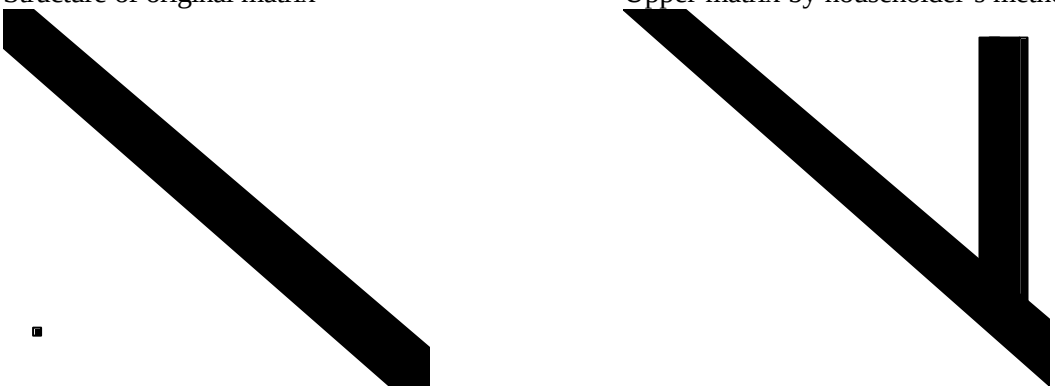


Figure 11: Fill-in for Householder's method for a band matrix with one element outside the band.

The amount of fill-in produced by only one element far from the diagonal can be seen in Figure 11.

4 SPARSE SYSTEMS OF NONLINEAR EQUATIONS

4.1 Introduction

If we want to find a solution to a nonlinear system of equations defined by

$$\mathbf{f}(\mathbf{x}) = 0, \tag{4.1}$$

we can, given an initial guess, iterate an approximate solution.
We will discuss two classes of methods for finding a solution to (4.1).
The first class of iterative methods for finding the solution to (4.1) is given by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{H}_k^{-1}\,\mathbf{f}\left(\mathbf{x}^{(k)}\right) \tag{4.2}$$

With $\mathbf{H}_k$ an approximation of $\frac{\partial}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x})$ in $\mathbf{x}^{(k)}$. Three choices of $\mathbf{H}_k$ are discussed.

- $\mathbf{H}_k = \mathbf{J}\left(\mathbf{x}^{(k)}\right)$ with $\mathbf{J}\left(\mathbf{x}^{(k)}\right)$ the Jacobian matrix of $\mathbf{f}(\mathbf{x})$ in $\mathbf{x}^{(k)}$. This method is called Newton's method.
- $\mathbf{H}_0 = \mathbf{J}\left(\mathbf{x}^{(0)}\right)$ and $\mathbf{H}_{k+1} = \mathbf{H}_k + \mathbf{v}\mathbf{w}^T$ update in the form of outer product (for uniquely defined **v** and **w**). This method is called Broyden's method.
- $\mathbf{H}_0 = \mathbf{J}\left(\mathbf{x}^{(0)}\right)$ and $\mathbf{H}_{k+1} = \mathbf{H}_k + \mathbf{B}_k$ with $\mathbf{B}_k$ a matrix of the same sparsity structure as $\mathbf{H}_k$ (with $\mathbf{B}_k$ uniquely defined). This method is called Schubert's method.

The second class of methods we will discuss are the continuation methods. We will discuss two methods which use (4.3) for solving (4.1).

$$\mathbf{h}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}) - (1-t)\,\mathbf{f}\left(\mathbf{x}^{(0)}\right). \tag{4.3}$$

The two methods are:
- Euler integration
- Newton homotopy

4.2 Newton's Method

Newton's method is one of the most common used methods for solving systems of nonlinear equations. In the neighborhood of a solution, Newton's method converges quadratically. If the Jacobian matrix cannot be computed exactly, and is computed by applying small perturbations around $\mathbf{x}^{(k)}$ the method converges super-linear. For every iteration of Newton's method we have to compute a Jacobian matrix and solve a system of linear equations.

4.3 Broyden's Method

In BROYDEN(1965) it is suggested to update the Jacobian matrix after every iteration instead of computing a new one. This update has the following properties:

$$\mathbf{H}_{k+1}\left(\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right)=\mathrm{f}\left(\mathbf{x}^{(k+1)}\right)-\mathrm{f}\left(\mathbf{x}^{(k)}\right) \tag{4.4}$$

$$\mathbf{H}_{k+1}\mathbf{z}=\mathbf{H}_{k}\mathbf{z},\ \text{whenever}\ \left(\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right)^{T}\mathbf{z}=0\,. \tag{4.5}$$

Properties (4.4) and (4.5) define the unique matrix

$$\mathbf{H}_{k+1}=\mathbf{H}_{k}+\frac{\left(\mathrm{f}\left(\mathbf{x}^{(k+1)}\right)-\mathrm{f}\left(\mathbf{x}^{(k)}\right)-\mathbf{H}_{k}\left(\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right)\right)}{\left|\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right|_{2}^{2}}\left(\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)}\right)^{T} \tag{4.6}$$

In MARWIL(1979) it is proved that Broyden's update defines the unique solution to

$$\min\left\{\left|\mathbf{H}_{k+1}-\mathbf{H}_{k}\right|_{\mathrm{F}}\ \middle|\ \mathbf{H}_{k+1}\ \text{satisfies (4.4)}\ \right\}\,, \tag{4.7}$$

were $|\cdot|_{\mathrm{F}}$ (APPENDIX A) denotes the Frobenius norm.
We define

$$\mathbf{p}_{k}=\mathbf{x}^{(k+1)}-\mathbf{x}^{(k)} \tag{4.8}$$

and

$$\mathbf{q}_{k}=\mathrm{f}\left(\mathbf{x}^{(k+1)}\right)-\mathrm{f}\left(\mathbf{x}^{(k)}\right). \tag{4.9}$$

The Sherman-Morrison formula (APPENDIX A) is applied to (4.6). We get

$$\mathbf{H}_{k+1}^{-1}=\mathbf{H}_{k}^{-1}+\frac{\left(\mathbf{p}_{k}-\mathbf{H}_{k}^{-1}\mathbf{q}_{k}\right)\mathbf{p}_{k}^{T}\mathbf{H}_{k}^{-1}}{\mathbf{p}_{k}^{T}\mathbf{H}_{k}^{-1}\mathbf{q}_{k}}. \tag{4.10}$$

If we apply Broyden's method for the solving of systems of nonlinear equations, we only have to compute the Jacobian or an approximation of the Jacobian matrix and its inverse at the first step. We can use (4.10) for computing the following steps of the iterative scheme defined by Equation (4.2). When the Jacobian matrix of $\mathrm{f}\left(\mathbf{x}\right)$ is very large and sparse, we can't use Equation (4.6) and (4.10) for two reasons: if we use the update defined by Equation (4.6) we add an outer-product of two vectors to the Jacobian. This outer-product produces in general a full matrix which is too large to store. For update defined by Equation (4.10) we need the inverse of the Jacobian which is in general not sparse either. Fortunately we can use Equation (4.10) in a slightly different way by computing the LU-decomposition and not the inverse. This method is described below.

If we define

(4.11) $$\mathbf{v}_k = \frac{\left(\mathbf{p}_k - \mathbf{H}_k^{-1}\mathbf{q}_k\right)}{\mathbf{p}_k^T \mathbf{H}_k^{-1}\mathbf{q}_k}$$

we can write (4.10) as

(4.12) $$\mathbf{H}_{k+1}^{-1} = \mathbf{H}_k^{-1} + \mathbf{v}_k \mathbf{p}_k^T \mathbf{H}_k^{-1}$$

At the $k+1^{st}$ step we want to compute

(4.13) $$\mathbf{H}_{k+1}^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right).$$

By (4.12) we have

(4.14) $$\mathbf{H}_{k+1}^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right) = \mathbf{H}_k^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right) + \mathbf{v}_k \mathbf{p}_k^T \mathbf{H}_k^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right),$$

$\mathbf{p}_k$ and $\mathbf{q}_k$ can be computed directly, but $\mathbf{v}_k$ cannot. We compute $\mathbf{v}_k$ by using the algorithm in Figure 12.

Compute $\mathbf{H}_0^{-1}\mathbf{v}_k$ by using the LU-decomposition of $\mathbf{H}_0^{-1}$
For i=1,..,k
$\quad \mathbf{H}_i^{-1}\mathbf{q}_k = \mathbf{H}_{i-1}^{-1}\mathbf{q}_k + \mathbf{v}_{i-1}\mathbf{p}_{i-1}^T \mathbf{H}_{i-1}^{-1}\mathbf{q}_k$
Endfor
compute $\mathbf{v}_k = \frac{\left(\mathbf{p}_k - \mathbf{H}_k^{-1}\mathbf{q}_k\right)}{\mathbf{p}_k^T \mathbf{H}_k^{-1}\mathbf{q}_k}$

Figure 12: Algorithm to compute $\mathbf{v}_k$.

By using a similar algorithm we can compute $\mathbf{H}_{k+1}^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right)$

compute $\mathbf{H}_0^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right)$ by using the LU-decomposition of $\mathbf{H}_0^{-1}$

For i=1,..,k+1 do step
$\quad \mathbf{H}_i^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right) = \mathbf{H}_{i-1}^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right) + \mathbf{v}_{i-1}\mathbf{p}_{i-1}^T \mathbf{H}_{i-1}^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right)$
Endfor

Figure 13: Algorithm to compute $\mathbf{H}_{k+1}^{-1}\, \mathrm{f}\left(\mathbf{x}^{(k+1)}\right)$.

There are two reasons why we limit the number of Broyden steps: The first reason is the recursive structure of the algorithm, every update takes more computations than its preceding step. A second

reason is that we have to store information from all earlier steps in the vectors $\mathbf{v}_i$ for i=1,..,n-1, $\mathbf{p}_i$ for i=1,..,n.
For applying the $k+1^{st}$ step of this Broyden's method for sparse systems we have to store from earlier steps the vectors $\mathbf{v}_i$ for i=1,..,n-1, $\mathbf{p}_i$ for i=1,..,n and $\mathbf{q}_k$. Whenever convergence is not reached after this limited number of Broyden steps we can restart Broyden's method using the current approximation as an initial guess.

4.4 Schubert's Method

Broyden's method for solving nonlinear equations generates a sequence of matrices which approximate the Jacobian matrix . Instead of property (4.5), SCHUBERT(1970) proposed to replace (4.5) by:

(4.15) $\mathbf{H}_{k+1}$ must have the same sparsity structure as $\mathbf{H}_k$.

Properties (4.4) and (4.15) do not define a unique matrix. Schubert proposed to choose the following update

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \sum_{\substack{j=1 \\ v_j \neq 0}}^{n} \frac{\mathbf{e}_j \mathbf{e}_i^{\mathrm{T}} \left(\mathbf{q}^{(k)} - \mathbf{H}_k \mathbf{p}^{(k)}\right) \mathbf{p}_j^{(k)^{\mathrm{T}}}}{\mathbf{p}_j^{(k)^{\mathrm{T}}} \mathbf{p}_j^{(k)}}$$

(4.16) $$\mathbf{p}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}, \quad \mathbf{q}^{(k)} = \mathrm{f}\left(\mathbf{x}^{(k+1)}\right) - \mathrm{f}\left(\mathbf{x}^{(k)}\right)$$

where $\mathbf{p}_j^{(k)} = \mathbf{D}_j \mathbf{p}^{(k)}$ and $\mathbf{e}_m$ denotes the m^{th} unit vector.

The matrix $\mathbf{D}_j$ represents the sparsity-structure of the j^{th} row of $\mathbf{H}_\mathrm{k}$

(4.17)
$$\mathbf{D}_j = \mathrm{diag}(d_1, d_2, \ldots, d_n)$$
$$d_i = \begin{cases} 1 & \text{if } h_{j,i} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

In MARWIL(1979) it is proved that Schubert's update (4.16) is the unique solution to

(4.18) $$\min\left\{ \left| \mathbf{H}_{k+1} - \mathbf{H}_k \right|_{\mathrm{F}} \middle| \mathbf{H}_{k+1} \text{ satisfies (4.4) and (4.15)} \right\}$$

For sparse matrices which have a special structure, Schubert's update converges in general faster than Broyden's update. The disadvantage of Schubert's method is that we can't use the Sherman-Morrisson formula, and therefore we have to solve a system of linear equations at every step.

4.5 Continuation Methods for Solving Systems of Nonlinear Equations

The iterative methods described in §4.2-4.4 only converge to the solution when the initial value $\mathbf{x}^{(0)}$ is chosen close to the solution $\mathbf{x}^*$ of (4.1). Continuation methods can be used to solve highly nonlinear equations and converge even if the initial guesses are bad. We will discuss continuation methods based on the following function

$$\mathrm{h}(\mathbf{x},t) = \mathrm{f}(\mathbf{x}) - (1-t)\,\mathrm{f}\left(\mathbf{x}^{(0)}\right), \tag{4.19}$$

where t is the homotopy parameter.
Euler integration and Newton homotopy are two methods to find a solution to equation (4.1) by using Equation (4.19).

4.5.1 Euler Integration

We can derive an ordinary differential Equation (ODE) from Equation (4.19) by defining the function $\mathbf{x}(t)$ such that $\mathrm{h}(\mathbf{x}(t),t) = \mathrm{f}(\mathbf{x}(t)) - (1-t)\,\mathrm{f}\left(\mathbf{x}^{(0)}\right) = 0$ for all t. A solution to Equation (4.1) equals $\mathbf{x}(1)$. If we differentiate $\mathrm{h}(\mathbf{x}(t),t)$ to t we get

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathrm{h}(\mathbf{x}(t),t) = \frac{\partial}{\partial \mathbf{x}}\mathrm{f}(\mathbf{x})\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) + \mathrm{f}\left(\mathbf{x}^{(0)}\right) = 0, \tag{4.20}$$

which leads to the ODE

$$\begin{cases} \dfrac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = -\mathbf{J}^{-1}(\mathbf{x}(t))\,\mathrm{f}\left(\mathbf{x}^{(0)}\right) \\ \mathbf{x}(0) = \mathbf{x}^{(0)} \end{cases}. \tag{4.21}$$

choose *steps*
set t=0
For k=1 to *steps*
t=t+1/*steps*
Compute $\mathbf{J}_{\mathrm{f}}\left(\mathbf{x}\left(\dfrac{k-1}{steps}\right)\right)$
Compute $\Delta\mathbf{x} = -\mathbf{J}_{\mathrm{f}}^{-1}\left(\mathbf{x}\left(\dfrac{k-1}{steps}\right)\right)\mathrm{f}\left(\mathbf{x}^{(0)}\right)$
$\mathbf{x}\left(\dfrac{k}{steps}\right) = \mathbf{x}\left(\dfrac{k-1}{steps}\right) + \dfrac{1}{steps}\Delta\mathbf{x}$
Endfor

Figure 14: The algorithm of Euler integration.

An approximation of $\mathbf{x}(1)$ is computed by applying Euler's method to the ODE (4.21). The algorithm is given in Figure 14.

4.5.2 Newton Homotopy

Newton Homotopy splits difficult problems up into a number of easy sub-problems which can be solved by using Newton's method. If we choose $0 = t_0 < t_1 < t_2 < \ldots < t_l = 1$ we get a number of problems to find the solution to $\mathrm{h}(\mathbf{x}, t_i) = 0$ for *i*=1,2..,*l*. The solution of $\mathrm{h}(\mathbf{x}, t_{i-1}) = 0$ is chosen as the initial guess for the solution of $\mathrm{h}(\mathbf{x}, t_i) = 0$. The solution to $\mathrm{h}(\mathbf{x}, t_j) = 0$ is close enough to the solution to $\mathrm{h}(\mathbf{x}, t_{j+1}) = 0$ for Newton's method to converge if the steps between t_j and t_{j+1} are chosen small. The algorithm of Newton homotopy is given in Figure 15.

Choose $0 = t_0 < t_1 < t_2 < \ldots < t_l = 1$
$\mathbf{x}_{\text{start}} = \mathbf{x}^{(0)}$
For k=1 to *l*
Use Newton's method to solve
$\mathrm{f}(\mathbf{x}) - (1 - t_k)\,\mathrm{f}(\mathbf{x}^{(0)})$
Initial guess: $\mathbf{x}_{\text{start}}$
Output: $\mathbf{x}_{\text{opl}}^{(k)}$
$\mathbf{x}_{\text{start}} = \mathbf{x}_{\text{opl}}^{(k)}$
Endfor
Output: $\mathbf{x}_{\text{opl}}^{(l)}$.

Figure 15: The algorithm of Newton Homotopy.

4.6 Damping

Newton's, Broyden's and Schubert's method for solving Equation (4.1) only converge to a solution if the initial guess $\mathbf{x}^{(0)}$ is close enough to this solution. To increase the domain of convergence we can replace (4.2) by

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - l^{(k)}\mathbf{H}_k^{-1}\,\mathrm{f}(\mathbf{x}^{(k)}), \qquad l^{(k)} \in (0,1] \tag{4.22}$$

There are several methods for choosing $l^{(k)}$. Whenever $0 < l^{(k)} < 1$ the method is called damped.

4.6.1 Standard Damping

The most applied way of damping is to choose for an arbitrary norm $l^{(k)}$ such that

(4.23) $$\left|\mathrm{f}\left(\mathbf{x}^{(k)} - l^{(k)}\mathbf{H}_k^{-1}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right)\right| < \left|\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right| \ .$$

Note that this does not imply $\left|\mathbf{x}^{(k)} - l^{(k)}\mathbf{H}_k^{-1}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right) - \mathbf{x}^{*}\right| < \left|\mathbf{x}^{(k)} - \mathbf{x}^{*}\right|$.

In most cases $l^{(k)}$ is chosen as

(4.24) $$l^{(k)} = \max\left\{q^{j}\,\middle|\, j \geq 0 \text{ and } \left|\mathrm{f}\left(\mathbf{x}^{(k)} - q^{j}\mathbf{H}_k^{-1}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right)\right| < \left|\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right|\right\} \text{ with } 0 < q < 1 \text{ fixed.}$$

It has be proven in practice that $q = 0.5$ or $q = 0.75$ are good choices.

4.6.2 Damping due to Deuflhard

The most usual way to choose the damping factor $l^{(k)}$ is by looking to the function value of the corresponding new approximation. In DEUFLHARD(1974) a different method for choosing $l^{(k)}$ is introduced. In each iteration we want to find a $l^{(k)}$ such that

(4.25) $$\left|\mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\left(\mathrm{f}\left(\mathbf{x}^{(k)} - l^{(k)}\mathbf{H}_k^{-1}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right)\right)\right| < \left|\mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right|$$

If Newton's method is applied then we obtain

(4.26) $$\mathbf{H}_k^{-1}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right) = \mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\mathrm{f}\left(\mathbf{x}^{(k)}\right)$$

This value has already been computed. $\mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\left(\mathrm{f}\left(\mathbf{x}^{(k)} - l^{(k)}\mathbf{H}_k^{-1}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right)\right)$ can be computed cheaply if $\mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\mathrm{f}\left(\mathbf{x}^{(k)}\right)$ is computed using the LU-decomposition. We can choose for example $l^{(k)} = \max\left\{q^{j}\,\middle|\, j \geq 0 \text{ and } \left|\mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\left(\mathrm{f}\left(\mathbf{x}^{(k)} - l^{(k)}\mathbf{H}_k^{-1}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right)\right)\right| < \left|\mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right|\right\}$ with $0 < q < 1$ fixed to find a suitable value for $l^{(k)}$.

As with standard damping we usually choose $q = 0.5$ or $q = 0.75$. By using this method less Newton steps are needed to find the solution of (4.1) than in the case of standard damping in practice but the amount of computations needed to find a suitable damping factor has increased.

4.6.3 Domain Damping

When dealing with actual problems we often have some additional information about the possible values of the solution in advance. For example, in chemical models negative concentrations are impossible. If we apply an iterative method to find the solution of such a problem, it is possible that the concentrations become negative. In this case we can have the following situations:

- Nothing special happens and after a number of iterations the concentrations become positive again.
- The iterative method converges to a non-realistic solution.
- The function is not defined for this value and the iterative method fails.
- The iterative method does not converge anymore.

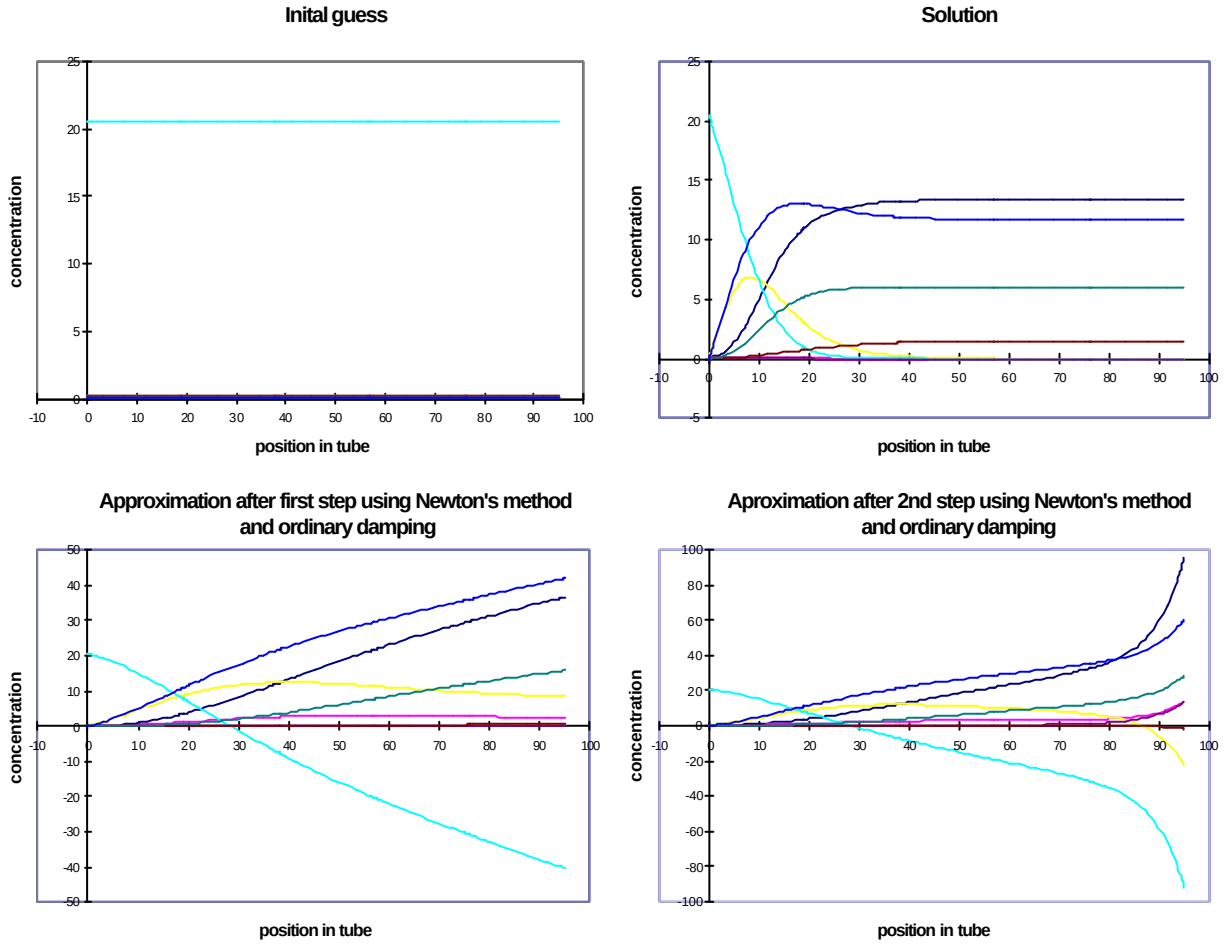


Figure 16: In this example Newtons method diverges. For this very stiff problem we cannot use ordinary damping methods but we have to use domain damping to find the solution.

Newton's method combined with the damping method as described in § 4.6 is used to solve a stiff ethylene model (Figure 16). After the first iteration, one of the components gets a negative concentration. Newton's method does not converge. In the third step, a value is computed for which the function is not defined and the iteration breaks down.

If something is known about the domain of the solution it is possible to find first the value $l_{\max}^{(k)}$ for which $\mathbf{x}^{(k)} - l_{\max}^{(k)} \mathbf{H}_k^{-1} \mathrm{f}\left(\mathbf{x}^{(k)}\right)$ is in the domain of realistic values. The damping factor can then be found in the domain $\left(0, l_{\max}^{(k)}\right]$.
It is possible that by a chosen initial value the method can only find a solution by leaving this domain of realistic values temporarily. For those types of problems this method can't be used.

4.6.4 Domain Damping with Reinitialization

Given the domain of realistic solutions it is possible that the solution lies on or close to the border of this domain. In this case the domain damping option of §4.6.3 can cause some problems for variables which lie close to the border and have converged faster than other variables. Due to small computational errors these variables can exceed the domain borders. In that case $l_{\max}^{(k)} = 0$ or $l_{\max}^{(k)} = l_{\min}$ is chosen. As an initialization tool and only if the function is defined for these variables in a small neighborhood outside the border we can apply domain damping on the domain $D \cup D_e$ where D_e denotes a small neighborhood around the border of *D*. All variables which have values exceeding *D* after this step of the iterative method are reinitialized on the border of *D*.

4.7 Scaling

A proper internal scaling plays a very important role for the efficiency and robustness of an algorithm. The purpose of scaling is to convert all variables to variables of approximately the same size. In this way we can reduce round-off errors. Especially additions become more accurate. The scaling transformation in $\mathbf{x}$ is defined by

$$\mathbf{x} \mapsto \mathbf{y} := \mathbf{S}_{\mathbf{x}}^{-1}\mathbf{x}, \tag{4.27}$$

where

$$\mathbf{S}_{\mathbf{x}} = \mathrm{diag}\left(s_{\mathbf{x}1}, s_{\mathbf{x}2}, \ldots, s_{\mathbf{x}n}\right). \tag{4.28}$$

Is a given non-singular diagonal transformation matrix. If we insert this transformation into the original problem we get a transformed problem

$$\mathrm{g}(\mathbf{y}) = 0, \tag{4.29}$$

with $\mathrm{g}(\mathbf{y}) = \mathrm{f}\left(\mathbf{S}_{\mathbf{x}}\mathbf{y}\right)$. The associated solution $\mathbf{y}^*$ and Jacobian matrix $\mathbf{J}_{\mathrm{g}}(\mathbf{y})$ are given by

$$\mathbf{y}^* = \mathbf{S}_{\mathbf{x}}^{-1}\mathbf{x}^*, \tag{4.30}$$

and

$$\mathbf{J}_{\mathrm{g}}(\mathbf{y}) = \frac{\partial}{\partial \mathbf{y}} \mathrm{f}\left(\mathbf{S}_{\mathbf{x}}\mathbf{y}\right) = \mathbf{J}_{\mathrm{f}}(\mathbf{x})\mathbf{S}_{\mathbf{x}}. \tag{4.31}$$

Similarly to scaling in $\mathbf{x}$ we can also scale in f. This transformation is defined by

(4.32) $$\mathrm{f}(\mathbf{x}) \mapsto \mathrm{h}(\mathbf{x}) := \mathbf{S}_\mathrm{f}^{-1}\,\mathrm{f}(\mathbf{x}),$$

with scaling matrix

(4.33) $$\mathbf{S}_\mathrm{f} = \mathrm{diag}(s_{\mathrm{f}1}, s_{\mathrm{f}2}, \ldots, s_{\mathrm{f}n}).$$

The associated Jacobian matrix $\mathbf{J}_\mathrm{h}(\mathbf{x})$ is given by

(4.34) $$\mathbf{J}_\mathrm{h}(\mathbf{x}) = \mathbf{S}_\mathrm{f}\,\frac{\partial}{\partial \mathbf{x}}\mathrm{f}(\mathbf{x}) = \mathbf{S}_\mathrm{f}\mathbf{J}_\mathrm{f}(\mathbf{x}).$$

The results of scaling for the various methods is discussed in the next paragraph.

4.8 Consequences of Scaling for Newton's, Broyden's and Schubert's Method

If we apply both scaling methods we get the transformed function

(4.35) $$\mathrm{h}(\mathbf{y}) = \mathbf{S}_\mathrm{f}\,\mathrm{f}(\mathbf{S}_\mathbf{x}\mathbf{y})$$

with $\mathbf{S}_\mathrm{f}$ and $\mathbf{S}_\mathbf{x}$ as defined in § 4.7. If we apply Newton's method on (4.35) we obtain

$$\begin{aligned}
\mathbf{y}^{(k+1)} &= \mathbf{y}^{(k)} - \mathbf{J}_\mathrm{h}\left(\mathbf{y}^{(k)}\right)^{-1}\mathrm{h}\left(\mathbf{y}^{(k)}\right) \\
&= \mathbf{y}^{(k)} - \left(\mathbf{S}_\mathrm{f}\mathbf{J}_\mathrm{f}\left(\mathbf{S}_\mathbf{x}\mathbf{y}^{(k)}\right)\mathbf{S}_\mathbf{x}\right)^{-1}\mathbf{S}_\mathrm{f}\,\mathrm{f}\left(\mathbf{S}_\mathbf{x}\mathbf{y}^{(k)}\right) \\
&= \mathbf{S}_\mathbf{x}^{-1}\mathbf{x}^{(k)} - \mathbf{S}_\mathbf{x}^{-1}\mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(k)}\right)^{-1}\mathbf{S}_\mathrm{f}^{-1}\mathbf{S}_\mathrm{f}\,\mathrm{f}\left(\mathbf{x}^{(k)}\right) \\
&= \mathbf{S}_\mathbf{x}^{-1}\left(\mathbf{x}^{(k)} - \mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(k)}\right)^{-1}\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right) \\
&= \mathbf{S}_\mathbf{x}^{-1}\mathbf{x}^{(k+1)},
\end{aligned}$$

with $\mathbf{J}_\mathrm{h}(\mathbf{y})$ the Jacobian matrix of $\mathrm{h}(\mathbf{y})$ in $\mathbf{y}$ and $\mathbf{J}_\mathrm{f}(\mathbf{x})$ the Jacobian matrix of $\mathrm{f}(\mathbf{x})$ in $\mathbf{x}$.
We see that Newton's method is invariant under scaling. This means that the path to the solution of the original problem equals the path of the scaled system.

If scaling is applied to Broyden's method, and we choose $\mathbf{H}_0 = \mathbf{S}_\mathrm{f}\mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(0)}\right)\mathbf{S}_\mathbf{x}$,

We find for the second step:

$$\begin{aligned}
\mathbf{y}^{(2)} &= \mathbf{y}^{(1)} - \frac{\left(\mathbf{y}^{(1)}-\mathbf{y}^{(0)}\right)-\left(\mathbf{S}_\mathrm{f}\mathbf{J}_\mathrm{f}\left(\mathbf{S}_\mathbf{x}\mathbf{y}^{(0)}\right)\mathbf{S}_\mathbf{x}\right)^{-1}\left(\mathrm{h}\left(\mathbf{y}^{(1)}\right)-\mathrm{h}\left(\mathbf{y}^{(0)}\right)\right)\left(\mathbf{y}^{(1)}-\mathbf{y}^{(0)}\right)^T\left(\mathbf{S}_\mathrm{f}\mathbf{J}_\mathrm{f}\left(\mathbf{S}_\mathbf{x}\mathbf{y}^{(0)}\right)\mathbf{S}_\mathbf{x}\right)^{-1}\mathrm{h}\left(\mathbf{y}^{(1)}\right)}{\left(\mathbf{y}^{(1)}-\mathbf{y}^{(0)}\right)\left(\mathbf{S}_\mathrm{f}\mathbf{J}_\mathrm{f}\left(\mathbf{S}_\mathbf{x}\mathbf{y}^{(k)}\right)\mathbf{S}_\mathbf{x}\right)^{-1}\left(\mathrm{h}\left(\mathbf{y}^{(1)}\right)-\mathrm{h}\left(\mathbf{y}^{(0)}\right)\right)} \\
&= \mathbf{S}_\mathbf{x}^{-1}\mathbf{x}^{(1)} - \frac{\mathbf{S}_\mathbf{x}^{-1}\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)-\mathbf{S}_\mathbf{x}^{-1}\mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(0)}\right)^{-1}\mathbf{S}_\mathrm{f}^{-1}\mathbf{S}_\mathrm{f}\left(\mathrm{f}\left(\mathbf{x}^{(1)}\right)-\mathrm{f}\left(\mathbf{x}^{(0)}\right)\right)\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)^T\mathbf{S}_\mathbf{x}^{-1^T}\mathbf{S}_\mathbf{x}^{-1}\mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(0)}\right)^{-1}\mathbf{S}_\mathrm{f}^{-1}\mathbf{S}_\mathrm{f}\,\mathrm{f}\left(\mathbf{x}^{(1)}\right)}{\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)^T\mathbf{S}_\mathbf{x}^{-1^T}\mathbf{S}_\mathbf{x}^{-1}\mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(0)}\right)^{-1}\mathbf{S}_\mathrm{f}^{-1}\mathbf{S}_\mathrm{f}\left(\mathrm{f}\left(\mathbf{x}^{(1)}\right)-\mathrm{f}\left(\mathbf{x}^{(0)}\right)\right)} \\
&= \mathbf{S}_\mathbf{x}^{-1}\mathbf{x}^{(1)} - \mathbf{S}_\mathbf{x}^{-1}\frac{\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)-\mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(0)}\right)^{-1}\left(\mathrm{f}\left(\mathbf{x}^{(1)}\right)-\mathrm{f}\left(\mathbf{x}^{(0)}\right)\right)\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)^T\mathbf{S}_\mathbf{x}^{-1^T}\mathbf{S}_\mathbf{x}^{-1}\mathbf{J}_\mathrm{f}\left(\mathbf{x}^{(0)}\right)^{-1}\mathrm{f}\left(\mathbf{x}^{(1)}\right)}{\left(\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\right)^T\mathbf{S}_\mathbf{x}^{-1^T}\mathbf{S}_\mathbf{x}^{-1}\left(\mathrm{f}\left(\mathbf{x}^{(1)}\right)-\mathrm{f}\left(\mathbf{x}^{(0)}\right)\right)}.
\end{aligned}$$

In general $\mathbf{S}_\mathbf{x}^{-1^T}\mathbf{S}_\mathbf{x}^{-1} \neq \mathbf{I}$, therefore Broyden's method is not scale invariant in $\mathbf{x}$.
Because Broyden's method is not scale invariant it is possible that the speed of convergence differs for different choices of $\mathbf{S}_\mathbf{x}$, even if all computations are exact.
Schubert's method, which is based on Broyden's method, is not scale invariant either.

5 THE IMPLEMENTATION

5.1 Introduction

In this chapter the implementation of the solver for nonlinear equations is discussed. The general structure of the solver consists of three blocks.

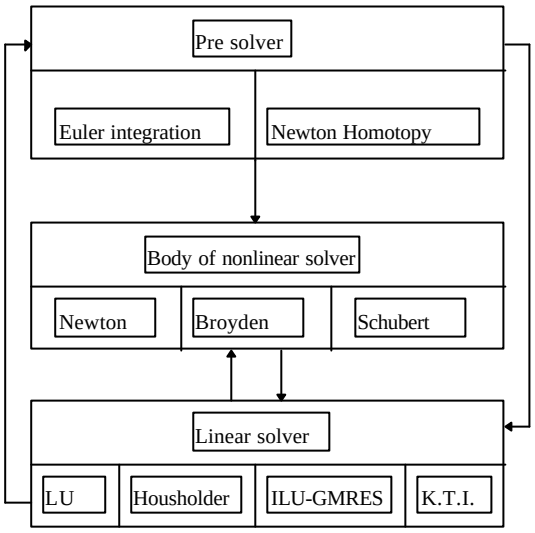


Figure 17: The general structure nonlinear solver.

The first block is the Pre-solver which produces an appropriate initial value. The second block is the main solver. This block consists of three methods for solving systems of nonlinear equations.
The third block is the linear solver. The linear solver is used by both the main solver and pre-solver.

5.2 The Linear Solver

There are 4 linear solvers available:

- KTI linear solver, which uses Gaussian elimination to compute the solution of the linear system.
- LU-decomposition algorithm, which can produce the solution or the LU-decomposition of the matrix.
- Householder's method which computes the solution of the linear system by orthonormal transformations.
- ILU-GMRES. A GMRES iterative method, preconditioned by a incomplete LU-decomposition.

5.2.1 The LU-decomposition algorithm

The LU-algorithm is a combination of LU-factorization with (partial) row pivoting and iterative refinement. To avoid fill-in due to small computational errors, a parameter $e > 0$ is introduced. For the computation of the Jacobian matrix a similar parameter is set. The elements of the lower matrix of the LU-decomposition consists of the quotients of the pivot element and the elements in the same column of the matrix during the decomposition procedure.

In the first step of the Gaussian elimination the smallest possible element in the lower matrix is bounded by the smallest element in the matrix divided by the largest element in the matrix. This 'local' idea is used as a indication for the maximal allowed size for e . In our problem the maximum elements in the Jacobian are of order 10^3 and the smallest elements of order 10^{-11} , $e = 10^{-20}$ is chosen as a tolerance. A second parameter is the real number *permtol* which is used to determine whether or not to permute two columns. At the *i*-th step of the elimination process, the columns *i* and *j* are permuted when $|a_{i,j}| \cdot permtol > |a_{i,i}|$.
The number of iterative refinements can also be set by a parameter.

5.2.2 The Householder Algorithm

If the Householder algorithm is applied to a matrix **A**, we compute **Q** such that $\mathbf{QA} = \mathbf{R}$, with **R** an upper triangular matrix and **Q** an unitary matrix. The unitary matrix **Q** is in general not sparse and can't be stored. For this reason we apply the same transformations on the right hand side of the linear system as on matrix **A**. When the algorithm is completed, we have reduced the system $\mathbf{Ax} = \mathbf{b}$ into the equivalent system $\mathbf{Rx} = \mathbf{Qb}$. The matrix **R** is upper triangular, hence this system can be solved directly by using back-substitution. The Householder algorithm needs columns of the matrix **A**. Therefore the algorithm is equipped with the sorting algorithm counting sort (APPENDIX B) .
If we use the SCOO or CRS format for internal computations, and the fill-in is produced as a result of operations on the matrix, we have to shift all the elements in the arrays storing the matrix.
The speed of the algorithm can be enhanced by using a variation of the CRS format for the internal computations. At the end of each column a fixed amount of space is left open. During the computations this space is used for storing the temporarily fill-in, hence no shifting of elements is needed.
Vectors are stored as sparse matrices. This storage is not primarily done to save memory but to speed up the computation time (we only look at non-zeros).
When applying the Householder algorithm we compute at every step

$$\mathbf{A}^{(k+1)} = \mathbf{A}^{(k)} - 2\mathbf{w}\mathbf{w}^{\mathrm{T}}\mathbf{A}^{(k)} \quad . \tag{5.1}$$

The second step is to compute $\mathbf{A}^{(k)} - 2\mathbf{w}\mathbf{p}^{\mathrm{T}}$ which equals $\mathbf{A}^{(k+1)}$. To avoid unnecessary fill-in due to small computation errors we only store elements in $\mathbf{A}^{(k+1)}$ if their absolute values are larger than a given e . As for the LU-decomposition $e = 10^{-20}$ is chosen.

5.2.3 The ILU GMRES Algorithm

If we apply the LU-decomposition algorithm on **A** we get **LU**=**A** (**L** is a lower triangular and **U** an upper triangular matrix). If no computational errors are made then **LU** is the most ideal preconditioning matrix. Instead of applying normal LU-decomposition we set some restrictions to the LU-decomposition algorithm. The first restriction is the maximum number of non-zero's allowed per row/column of the two factorization matrices **L** and **U**. The second restriction is to set a minimum for the absolute value of the elements in the factorization matrices **L** and **U**. By using these two restrictions we get

$$\mathbf{A} \approx \tilde{\mathbf{L}}\tilde{\mathbf{U}} \tag{5.2}$$

$\tilde{\mathbf{L}}\tilde{\mathbf{U}}$ are the matrices computed using incomplete LU-decomposition. The advantage of using incomplete LU-decomposition is that the maximum amount of fill-in can be controlled by setting the maximum number of non-zero's per row. The element restriction parameter e is chosen such that all the available places in the LU-decomposition are used by the large 'important' elements.
The GMRES algorithm solves the preconditioned linear system. The convergence criteria must be given and the maximum size of the Krylov space must be set.

5.3 Initialization with Euler Integration or Newton Homotopy

The pre-solvers are implemented straight forward from the algorithms given in §4.5.1 and §4.5.2. The number of steps must be given for both Euler integration and Newton homotopy. For the Newton homotopy the maximum number of Newton iterations per step can be set and the convergence criteria for the homotopy steps must be given. Damping methods can be used but for a robust pre-solving, the number of steps must be chosen large enough to make damping unnecessary. Scaling is not implemented in either algorithms.

5.4 The Body of the Solver for Nonlinear Equations

In the body of the solver there are three methods implemented for solving systems of nonlinear equations.
These methods are:

- Newton's method
- Broyden's method
- Schubert's method

In the main solver the scaling is updated at every step. Stopping criteria are tested and the new Jacobian is computed.

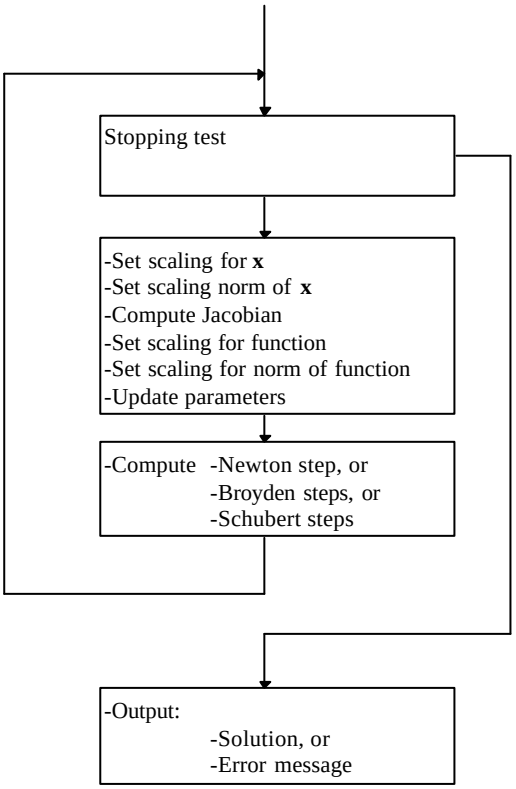


Figure 18: Structure body of nonlinear solver.

It is possible to update parameters during the iterations. This option is for example used to switch off the domain reinitialization when convergence has almost been reached.

5.4.1 Stopping Test

The system of nonlinear equations is solved by determining an approximate value $\tilde{\mathbf{x}}$ for the solution $\mathbf{x}^*$ that complies with a given tolerance.
There are three termination criteria for an iterative solver:

- stopping criteria depending on the value of the approximation $\tilde{\mathbf{x}}$

(5.3)
$$\left|\tilde{\mathbf{x}}-\mathbf{x}^*\right| \le e_{\mathbf{x}}, \text{ or the relative error}$$
$$\left|\tilde{\mathbf{x}}-\mathbf{x}^*\right| \le e_{\mathbf{x}}\left|\mathbf{x}^*\right|$$

- The value of the function at the approximation

(5.4)
$$\left|\mathrm{f}(\tilde{\mathbf{x}})\right| \le e_{\mathrm{f}}$$

- There is no solution or the solution cannot be found by this iterative method.

The value of $\mathbf{x}^*$ is in general not known hence we cannot use (5.3). If we define:

(5.5)
$$\Delta x^{(k)} = \text{correction of iteration (without damping)},$$

we can use

(5.6)
$$\left|\Delta\mathbf{x}^{(k)}\right| \le e_{\mathbf{x}} \text{ or } \left|\Delta\mathbf{x}^{(k)}\right| \le e_{\mathbf{x}}\left|\mathbf{x}^{(k)}\right|$$

instead of (5.3),
but if convergence is slow this can lead to a premature termination. Therefore both methods (5.6) and (5.4) are used.
In general relative errors are very useful. However in chemical engineering problems are usually badly scaled. Relative norms can't be used as a 'fair' stopping criteria. Instead of relative norms we use scaled norms.
The scaled norm is defined by

(5.7)
$$\left|\mathbf{y}\right|_{sc} = \left|\mathbf{D}\mathbf{y}\right|_2$$

with **D** the scaling diagonal matrix.

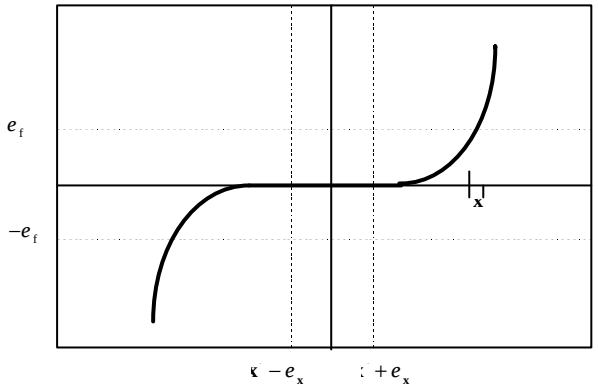


Figure 19: Termination on function value: Iterations are stopped but convergence in x is not yet reached.

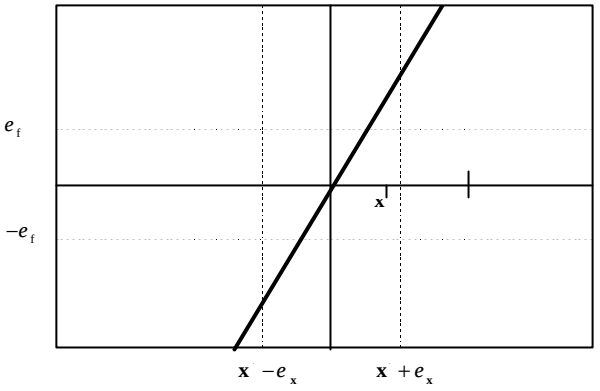


Figure 20: Termination on function value: Iterations continue while convergence in x is already reached.

The drawback of using criterion (5.4) is illustrated in Figure 19 and Figure 20. In Figure 19 the iteration is terminated before convergence in **x** is reached. In Figure 20 the convergence criterion in **x** is already reached but criterion (5.4) not. This results in unnecessary iterations.
This problem can in many cases be solved by using a dual stopping criterion. The stopping criteria used in the nonlinear solver are:
Stop iterations when

- $\left|\Delta \mathbf{x}^{(k)}\right|_{scx} \le e_{\mathbf{x}}$ and $\left|\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right|_{sc\,\mathrm{f}} \le e_{\mathrm{f}}$

  or,
- maximum number of iterations exceeded.

The choice of the two scaling matrices are:

$$\mathbf{D}_{\mathrm{f}} = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix}, \text{ with } d_k = \left(\sum_{l=1}^{n} \left| j_{k,l} \right| \right)^{-1} \tag{5.8}$$

with $j_{\mathrm{i,j}}$ the elements of the last computed Jacobian matrix. This choice is based on the fact that $\mathbf{D}_{\mathrm{f}}$ is a measure of the steepness of $\mathrm{f}\left(\mathbf{x}^{(k)}\right)$. And

$$\mathbf{D}_{\mathbf{x}} = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix}, \text{ with } d_i = \begin{cases} \left| \mathbf{x}_i^{(k)} \right| & \text{if } \left| \mathbf{x}_i^{(k)} \right| \le e_{tol} \\ e_{tol}, & \text{otherwise} \end{cases} \tag{5.9}$$

The scaled norm in **x** can be seen as a component wise relative norm.

5.4.2 Damping Methods

There are three different methods implemented for the sequence in which the damping factors are chosen.

- Safe damping:
  The sequence of damping factors is given by $l_{\,l} = q^{l}$ with $0 < q < 1$ and $l = l_{\max}, l_{\max} - 1, \ldots, 1, 0$.
- Standard damping
  The sequence of damping factors is given by $l_{\,l} = q^{l}$ with $0 < q < 1$ and $l = 0, 1, \ldots, l_{\max} - 1, l_{\max}$.
- Domain damping
  The sequence of damping factors is given by $l_{\,l} = q^{l}$ with $0 < q < 1$ and $l = 0, 1, \ldots, l_{\max} - 1, l_{\max}$
  Damping criteria are checked only when the function is defined for $\mathbf{x}^{(k)} + l_l^{(k)} \Delta \mathbf{x}^{(k)}$.

A damping factor is accepted whenever the next damping factor in the sequence is worst than the current factor.
When the nonlinear function is a 'black box' and no information about the domain of definition is known we have to be very careful when choosing a damping factor. If $\mathrm{f}\left(\mathbf{x}^{(k)} + \Delta \mathbf{x}^{(k)}\right)$ is not defined then checking the damping criteria can result in a premature break down of the iterative process. Safe damping needs more computations but is much safer than standard damping because a suitable damping factor will already be found before the domain of definition is exceeded. Safe damping is not completely safe and can result in a breakdown of the iterations, for example when $\mathrm{f}\left(\mathbf{x}^{(k)} + l_l^{(k)} \Delta \mathbf{x}^{(k)}\right)$ is defined and $\mathrm{f}\left(\mathbf{x}^{(k)} + l_{l-1}^{(k)} \Delta \mathbf{x}^{(k)}\right)$ not.
Domain damping is the safest method but it requires a domain of the variables.

5.4.3 Newton's Method

There are two separate subroutines for Newton's method. The differences between the two subroutines are the available damping methods. The first one supports the most common method for damping, which is based on the function values of the approximations $\mathbf{x}^{(k)} + l_k \Delta\mathbf{x}^{(k)}$. For this method all linear solvers can be used. The second method due to DEUFLHARD(1973) uses a modified Newton step at $\mathbf{x}^{(k)} + l_k \Delta\mathbf{x}^{(k)}$ to determine the damping factor. This method can only be used in combination with the LU-factorization algorithm for solving systems of linear equations. The LU-decomposition must be saved for the damping procedure and the linear solver is an integral part of the algorithm.
The implementation of Newton's method is illustrated in Figure 21 and Figure 22.

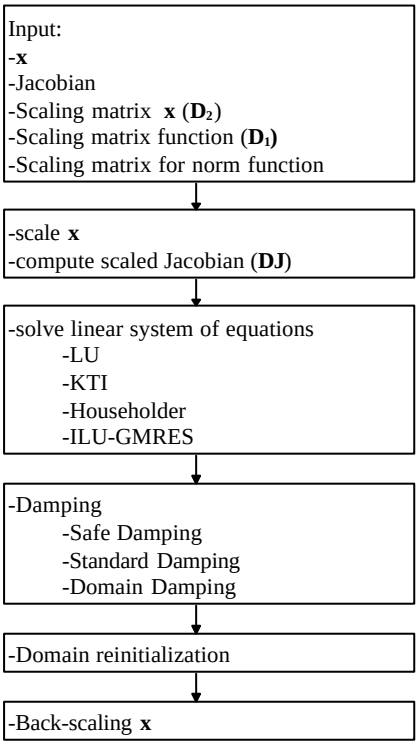


Figure 21: The algorithm of Newton's method using usual damping techniques.

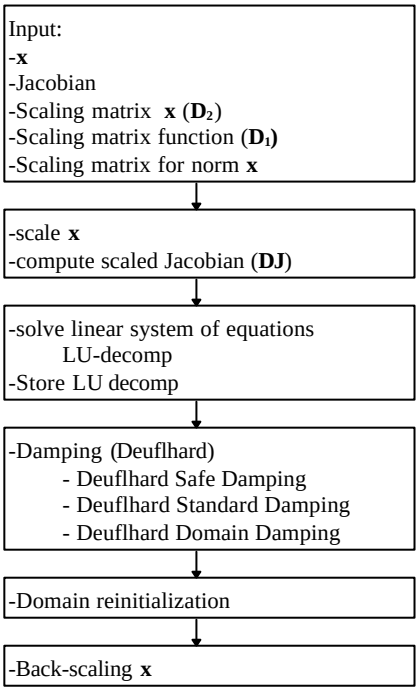


Figure 22: The algorithm of Newton's method using Deuflhard damping.

5.4.4 Broyden's Method

Broyden's method computes one Newton-step followed by a number of Broyden steps. The recursive procedure to compute the updates becomes more costly, and more information has to be stored every time. Therefore the maximum number of iterations is limited. This can be set by a parameter. Another feature of the algorithm is the possibility to continue the updates even when $\left|\mathrm{f}\left(\mathbf{x}^{(k)}\right)\right|_{scf} < \left|\mathrm{f}\left(\mathbf{x}^{(k+1)}\right)\right|_{scf}$. In this case the value of $\mathbf{x}^{(k)}$ is stored as $\mathbf{x}_{\mathrm{opt}}$ and when $\left|\mathrm{f}\left(\mathbf{x}_{\mathrm{opt}}\right)\right|_{scf} < \left|\mathrm{f}\left(\mathbf{x}^{(\mathrm{final})}\right)\right|_{scf}$, the algorithm returns $\mathbf{x}_{\mathrm{opt}}$. To avoid the possibility of a break down due to an exceeding divergence, the updates are

stopped when $maxdiv \cdot \left|\mathrm{f}\left(\mathbf{x}_{\mathrm{opt}}\right)\right|_{scf} < \left|\mathrm{f}\left(\mathbf{x}^{(m)}\right)\right|_{scf}$. The default value of *maxdiv* equals 1.0. This implicates that the updates are stopped immediately if no improvement is made. For applying the recursive update we need an LU-decomposition, therefore we have to use the LU-solver. The implementation of Broyden's method is illustrated in Figure 23.

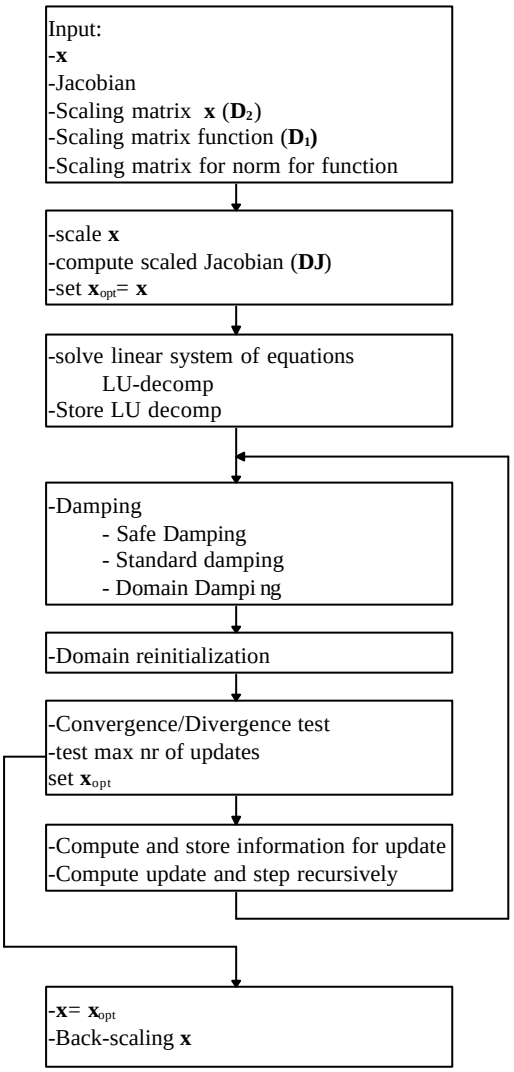


Figure 23: The algorithm of Broyden's method.

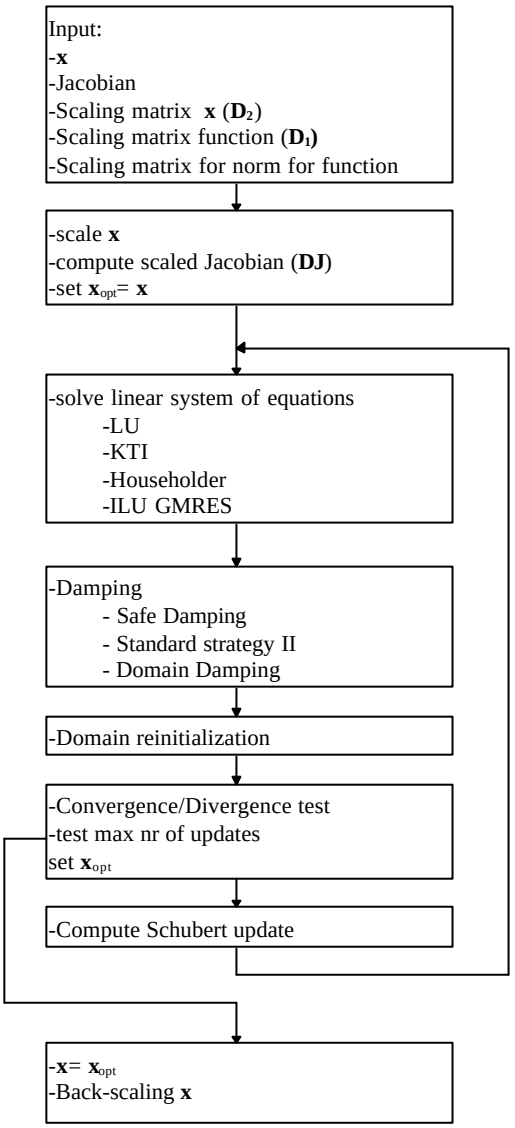


Figure 24: The algorithm of Schubert's Method.

5.4.5 Schubert's Method

Schubert's method is similar to Broyden's method. The only difference is that instead of the recursive update an explicit update is computed. Therefore it is also possible to compute the linear system by all linear solvers. The implementation of Broyden's method is illustrated in Figure 24.

6 TEST RESULTS

6.1 Introduction

The various parts of the solver for sparse systems of nonlinear equations, which are described in Chapter 5, were tested and the results are reported in this chapter. Due to the number of settings of the solver and different models it is impossible to present all test information in this chapter. Therefore a limited number of representative test results are given.
First the methods for solving linear systems of equations are tested. The test results of direct methods for solving linear systems of equations are presented in §6.2.1. The results of the ILU-GMRES algorithm are presented in §6.2.2. These tests results contain information on accuracy and speed of the various methods.
The number of iterations needed for Newton's method to converge depends on the used damping methods. In §6.3 results are presented of Newton's method using different damping methods. Broyden's method and Schubert's method are not scale-invariant. These two methods are tested and results are presented in §6.4. For stiff models we have to use different techniques to find a solution. The results of continuation methods and the domain damping with reinitialization for solving a stiff model are given in §6.5. To avoid unnecessary iterations, we have to find a proper stopping criterion. In §6.6 we use scaled norms to stop iterating at the right time. Because the SPYRO equations are not available at this moment, a problem of approximately the same size as SPYRO has been created. To estimate the computational time for solving the SPYRO model, this model is solved and the results are presented in §6.7
The initial value to solve the steam cracker model is a horizontal profile. This is illustrated in Figure 25.

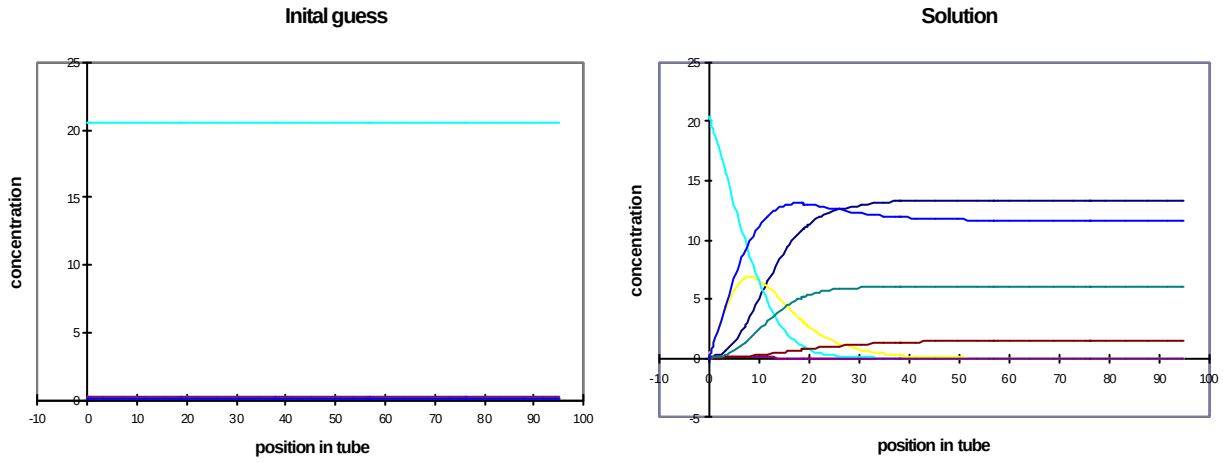


Figure 25: The initial guess and the final solution of the stiff model PER (stiffness factor=100). The initial guess for all the tested models equals the initial guess as shown in this figure.

6.2 Linear Solvers

First we will determine which linear solver to use for the algorithms to solve systems of nonlinear equations.
Three direct linear solvers have been tested. These solvers are:

- the LU-decomposition algorithm,
- the KTI linear solver[1],
- Householder's method and
- ILU-GMRES (this is an iterative method).

To test the linear solvers we have to construct a system of linear equations of which the solution is known. If the matrix is given, it is possible to choose a vector **y** and define

(6.1) $$\mathbf{b} = \mathbf{A}\mathbf{y}$$

We can now solve the system

(6.2) $$\mathbf{A}\mathbf{x} = \mathbf{b}$$

The solution of this system is known. The computed solution of (6.2) can be now be compared to the exact solution **y**. Note: it is assumed that **Ay** is computed exactly.
The choice of **y** will be described below.
The test problems for the linear solvers are constructed from the first step of Newton's method of the nonlinear system of equations. If we define

(6.3) $$\mathbf{c} = -\mathrm{f}(\mathbf{x}),$$

then we have to find the solution of

(6.4) $$\mathbf{J}\mathbf{y} = \mathbf{c}$$

We will now construct from (6.4) a system of linear equations of which the solution is known. First we compute an approximation of the solution of (6.4) by using one of the methods for solving the linear systems of equations. This approximation $\tilde{\mathbf{y}}$ will be the solution of the constructed test problem.
The solvers for systems of linear equations will be tested for the system

(6.5) $$\mathbf{J}\mathbf{x} = \left(\mathbf{J}\tilde{\mathbf{y}}\right)$$

We will first discuss the test results of the direct methods.

6.2.1 Direct Linear Solvers

We will use the first Newton iteration of the following systems to test the direct methods for solving sparse systems of linear equations:

- KR1, 6 collocation points and 6 sections, dimension=467
- KR3, 6 collocation points and 19 sections, dimension=1265

[1] The KTI linear solver is a undocumented in-house solver for sparse systems on linear equations.

- KR5, 6 collocation points and 6 sections, dimension=467
- KR6, 6 collocation points and 6 sections, dimension=467
- KR7, 6 collocation points and 6 sections, dimension=43051
- KR8, 6 collocation points and 40 sections, dimension=3051

The tolerance for dropping small elements is set to 10^{-20} , both for the LU-decomposition as well as for Householder's method. The permutation tolerance for the LU-decomposition is set to 10^{-2} as suggested in SAAD(1994) and DUFF(1986).

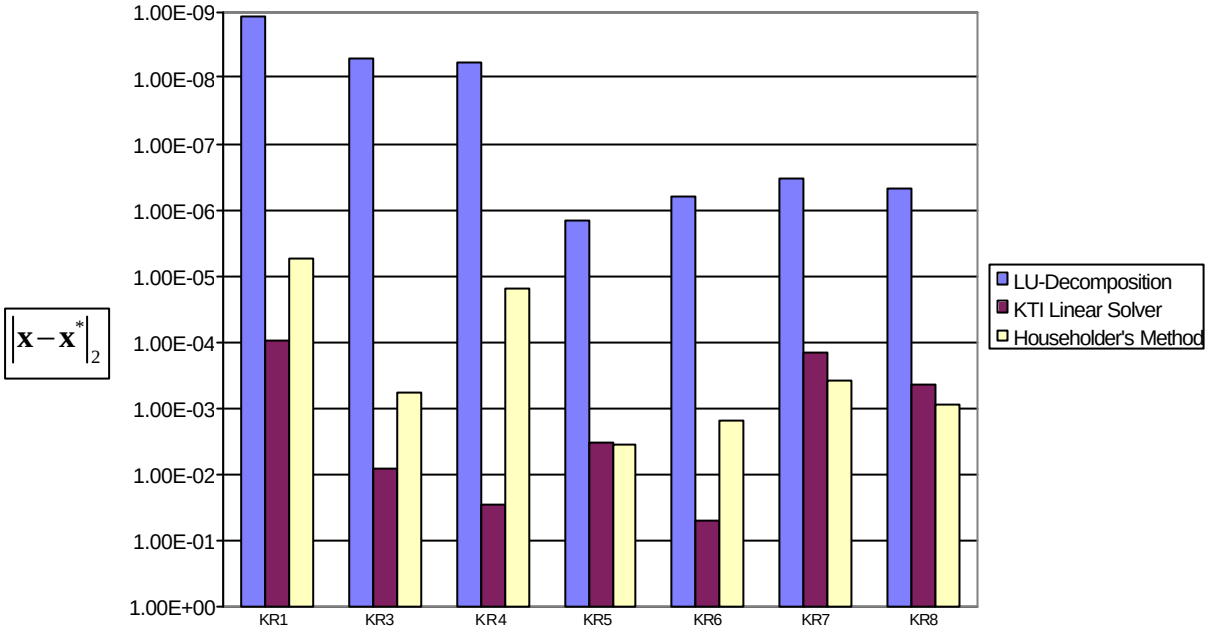


Figure 26: The accuracy of the tested direct method for linear systems. The solution computed by the LU-decomposition is significally better than the solutions computed by Householder's method and the KTI solver.

The accuracy of the methods are illustrated in Figure 26. In all tests the LU-decomposition computes the best approximation of the solution.

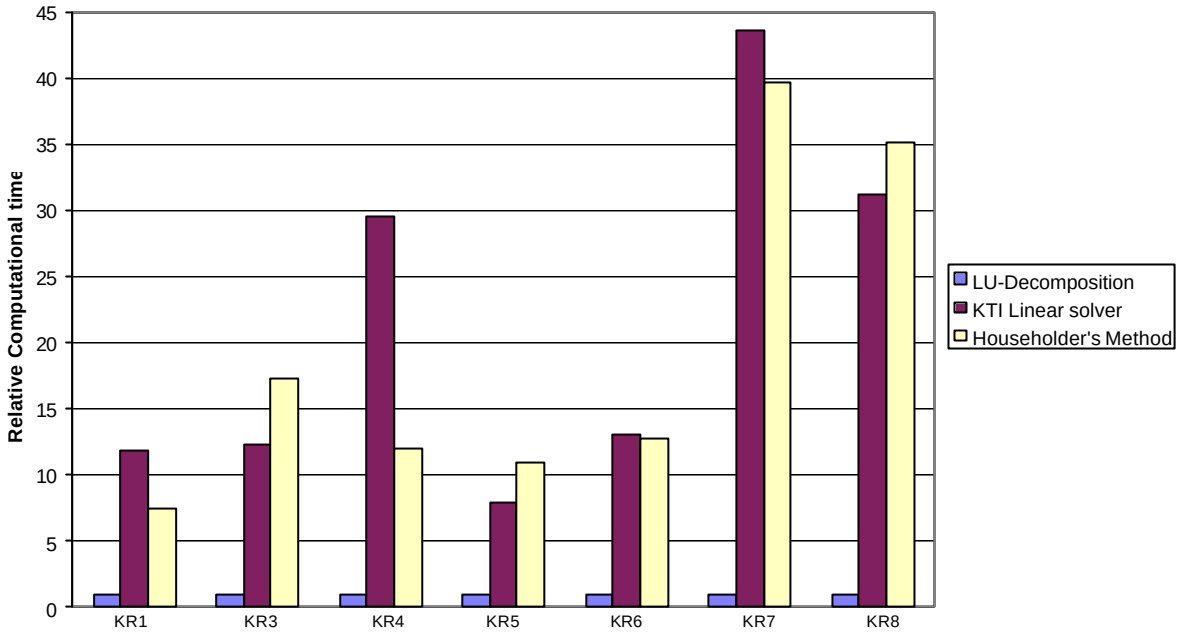


Figure 27: The Computational time of the direct linear solvers related to the LU-decompostion algorithm. The LU-decomposition is the fasted method for all the tested problems.

To construct a fast solver for systems of nonlinear equations, we need a fast solver for linear systems of equations. In Figure 27 the computational time is given for the three solvers. The computation time is related per system to the computation time of the LU-decomposition.

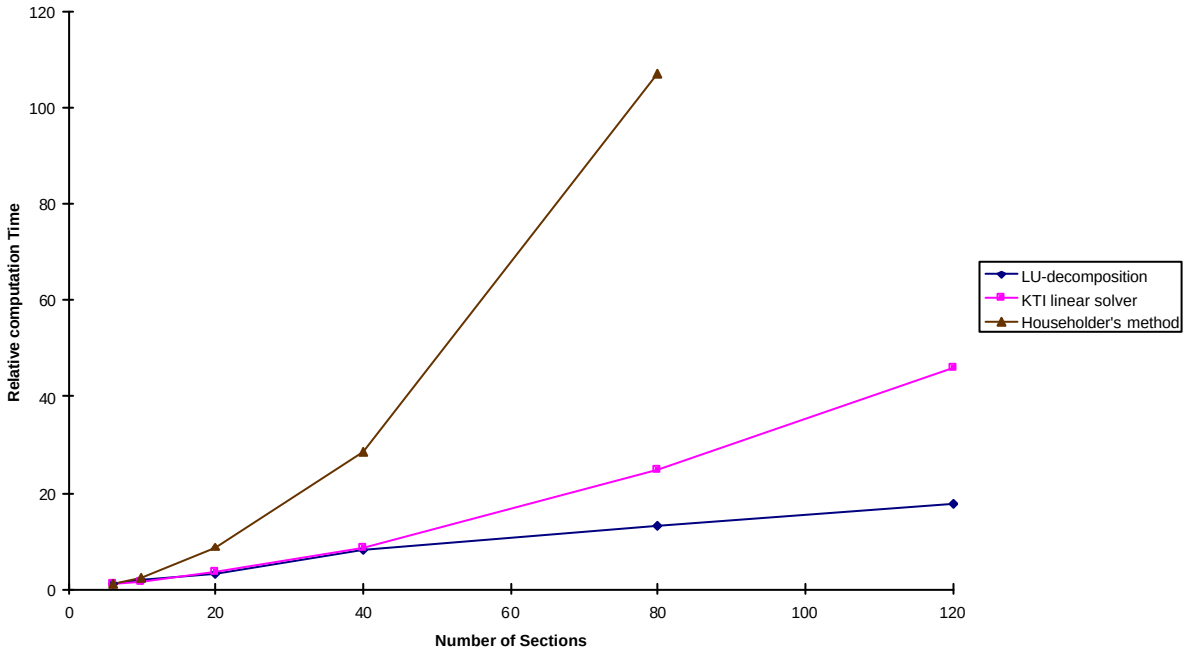


Figure 28: The relative computational time for the direct linear solvers for KR1 with various numbers of sections. The LU-decomposition are related linear with the number of sections. Householder's method however is related superlinear.

For the steam cracker models we expect that the computation time is linear dependent on the number of sections. This has been tested using model KR1 with several numbers of sections. The relative computational time is presented in Figure 28. For the KTI linear solver and the LU-decomposition we can see the linear dependency in Figure 28. Householder's method however appears to have a super-linear dependency.

The LU-decomposition is the fastest and most accurate of tested direct solvers for sparse systems of linear equations. Another advantage is that the LU-decomposition can be stored, and can be used later if a system of linear equations with the same coefficient matrix has to be solved.

6.2.2 The Iterative Linear Solver

We will consider the ILU-GMRES algorithm to solve systems of linear equations iterative. The initial guess for the solution of the linear systems is the zero-vector.
Model KR1 with 6 collocation points, 10 sections and a dimension of 641 is used to test the ILU-GMRES algorithm.
The iterations stop whenever $\left|\mathbf{Ax}^{(i)}\right| / |\mathbf{b}| < 10^{-25}$. In the presented tests the maximum size of the Krylov space is set to 10.
If no preconditioning is used the GMRES algorithm does not converge to the solution. The ILU-GMRES algorithm is preconditioned by an incomplete LU-decomposition. For the incomplete LU-decomposition the number of elements in each row/column and the minimum size of elements in the LU-decomposition can be set. For the first test we restrict the number of elements in each row/column of the incomplete LU-decomposition. The computational time in relation to the LU-decomposition is given in Figure 29. If the maximum size is set to 15, the ILU-GMRES algorithm does not converge within 500 iterations.

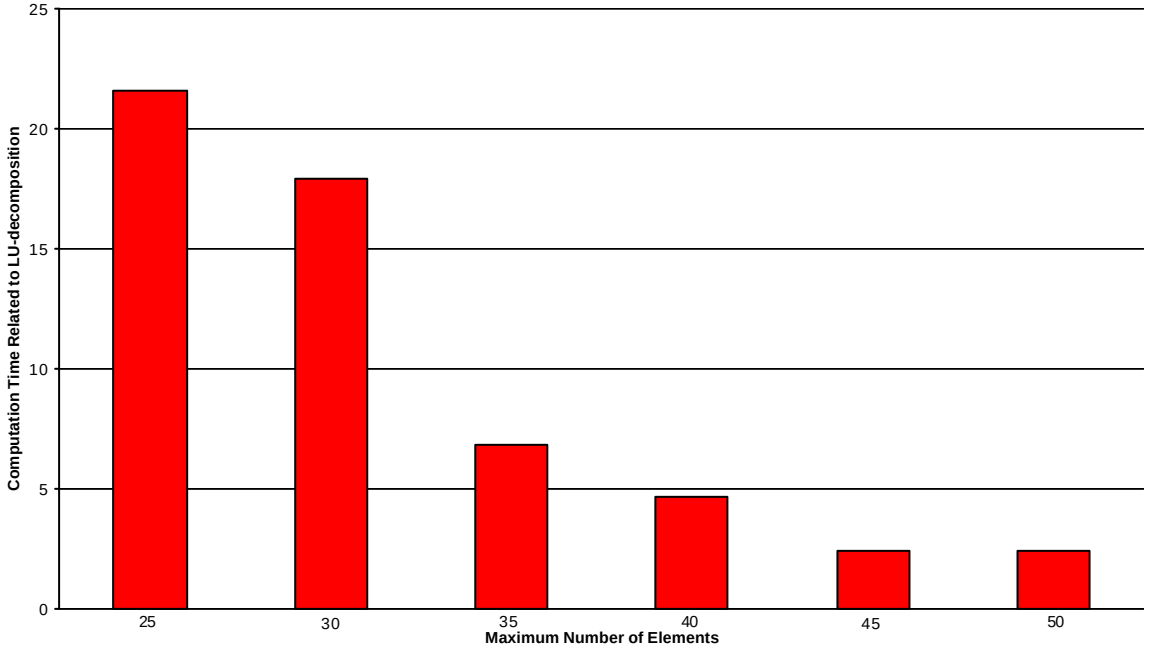


Figure 29: The relative computational time for the ILU-GMRES algorithm. The number of elements in the preconditioning matrix are restricted.
We see in Figure 29 that the ILU-GMRES algorithm is not faster than the LU-decomposition. The number of iterations and the achieved precision are presented in Figure 30.

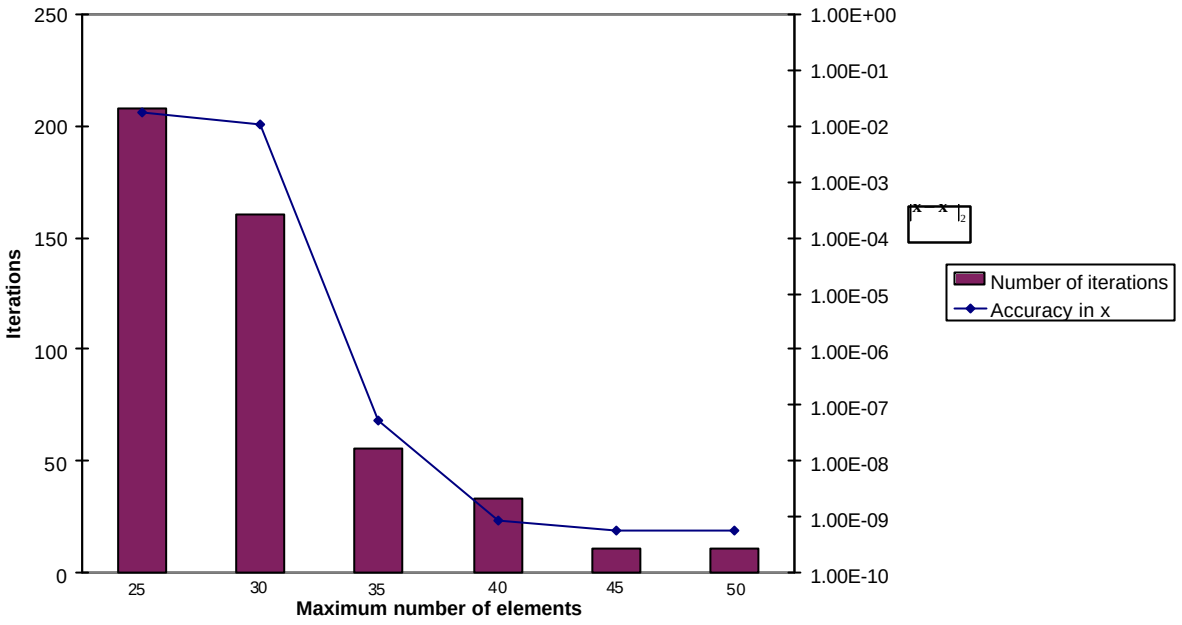


Figure 30: The Number of iterations and accuracy of the final solution of the ILU_GMRES algorithm with a restricted number of elements in the preconditioning matrix

The results for a restriction of the absolute value of the elements in the incomplete LU-decomposition are presented in Figure 31 and Figure 32.

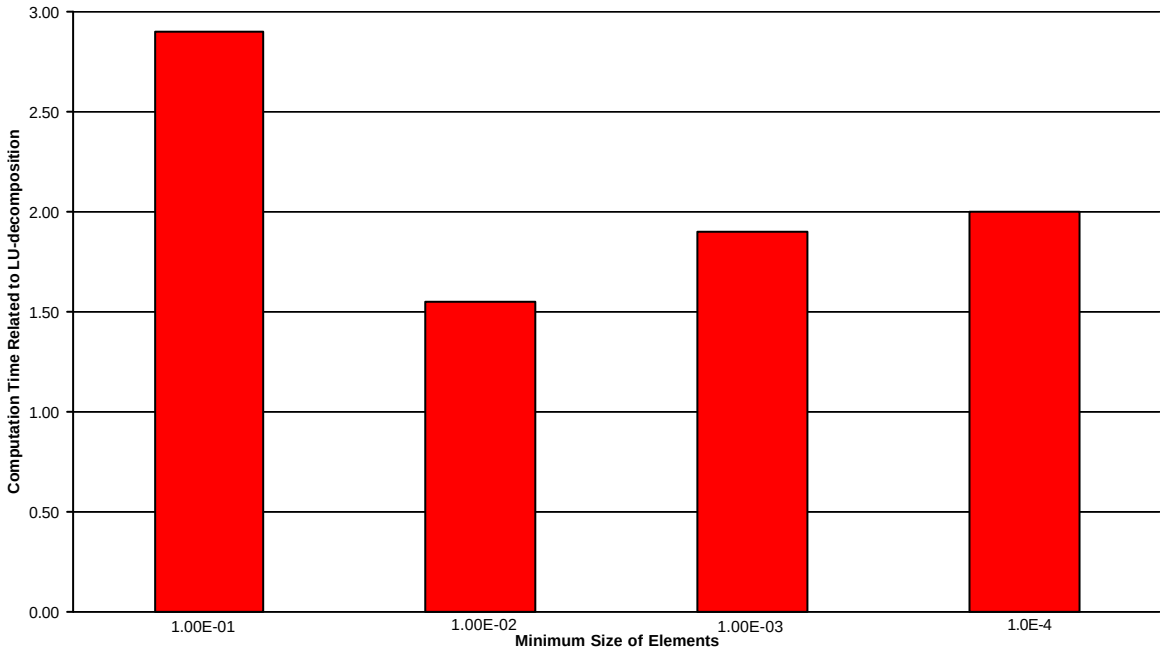


Figure 31: The relative computational time for the ILU-GMRES algorithm for a limited absolute value of the elements in the preconditioning matrix

If the minimum absolute value of the elements in the incomplete LU-decomposition is set to 1.0, the ILU-GMRES algorithm does not converge.

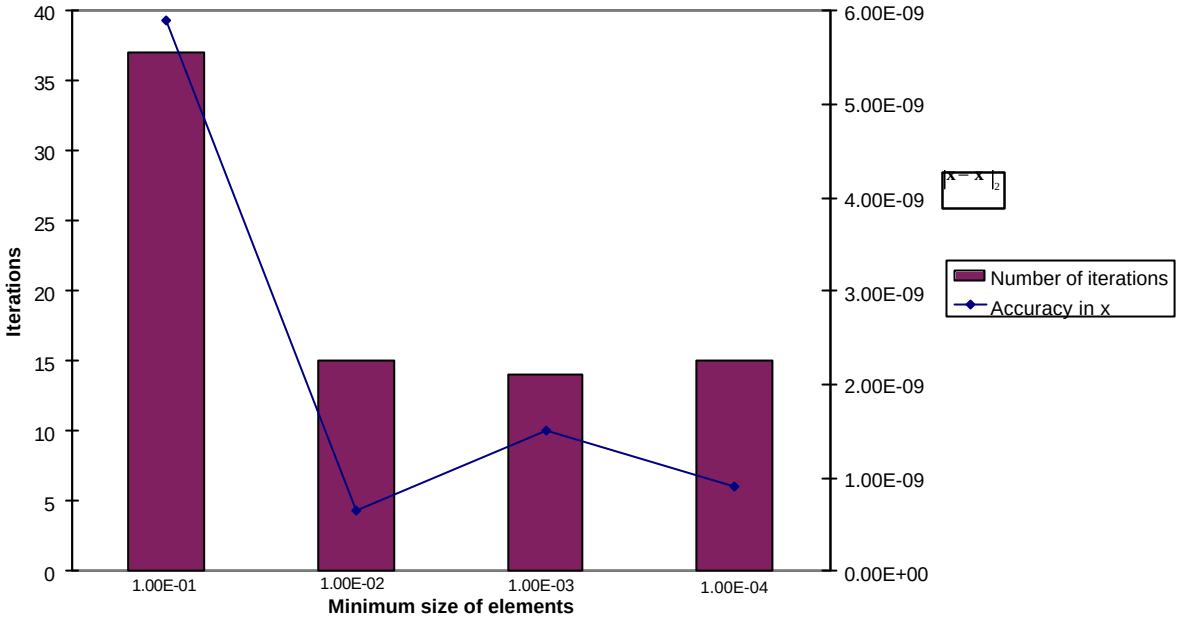


Figure 32:The Number of iterations and accuracy of the final solution of the ILU-GMRES algorithm with a restriction on the absolute value of the elements in the preconditioning matrix

We need a very good preconditioning matrix for the GMRES to converge. And if we look at the accuracy of the results we see in Figure 30 that it varies between the order of 10^{-2} and 10^{-10}. This is not to be expected for the convergence criterion. COFER(1996) obtained the same results when testing iterative methods for solving linear systems evolving from chemical simulation processes. The reason of this behavior is that chemical problems are in general poorly conditioned. The LU-decomposition algorithm is in all tested cases faster than the preconditioned GMRES algorithm.

6.3 Newton's Method

If we use Newton's method to solve systems of nonlinear equations we have to compute the Jacobian matrix and solve a system of linear equations at every iteration. The differences between the computation time of the different damping methods can be neglected. Therefore we only compare the number of iterations needed to converge. We want to find a damping method that minimizes the number of iterations.
Newton's method combined with various methods for damping as described in §4.6.1, §4.6.2 and §5.4.2 is tested on all available models. The iterations are stopped when $\left|\mathbf{f}(\mathbf{x})\right|_2 < 10^{-7}$ .The results are given in Figure 33.

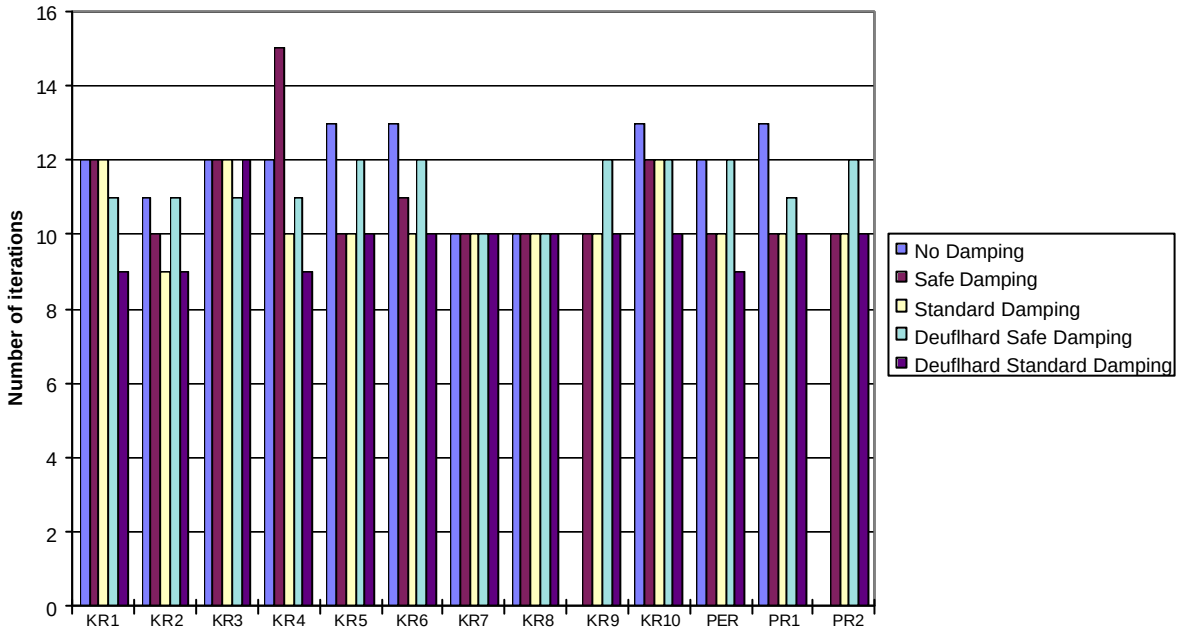


Figure 33: The convergence of Newton's method using various damping methods for all available test models.

Model KR9 and PR2 do not converge if no damping is used.
If damping methods are used all models converge and less iterations are needed than without damping. The only exception is safe damping for solving model KR4 where three extra iterations are needed. For all models except KR3 the fastest convergence is obtained using Deuflhard standard damping. Therefore if Newton's method is applied to solve systems of nonlinear equations, we will use standard Deuflhard damping.
Different types of scaling were tested but scaling did not influence the performance of Newton's method for the tested problems. This can be explained by the fact that the linear system is computed accurately even if no scaling is used, and Newton's method itself is scale invariant.

6.4 Broyden's & Schubert's Method

Broyden's method and Schubert's method both update the Jacobian matrix instead of computing a new one. For Schubert's method we have used the LU-decomposition to solve linear systems of equations. We compare the speed of convergence between Broyden's method and Schubert's method by solving the models

- KR4 with 100 sections and 7 collocation points per section(dim=8311).
- KR8 with 40 sections and 6 collocation points per section(dim=3051).

Deuflhard damping can't be used because this method needs the LU-decomposition of the current Broyden iteration and this Jacobian is not known. By testing it is found that normal damping can cause some problems (if the 'damping' factor $l^{(k)} = 1$ is used to check the damping criterion for a Broyden iteration). Therefore we use safe damping for Broyden's method.
For both models we use safe damping. The stopping criterion is $\left|\mathbf{f}(\mathbf{x})\right|_2 < 10^{-7}$

To check which Jacobian update is to be preferred, Broyden or Schubert, we will consider the number of iterations.

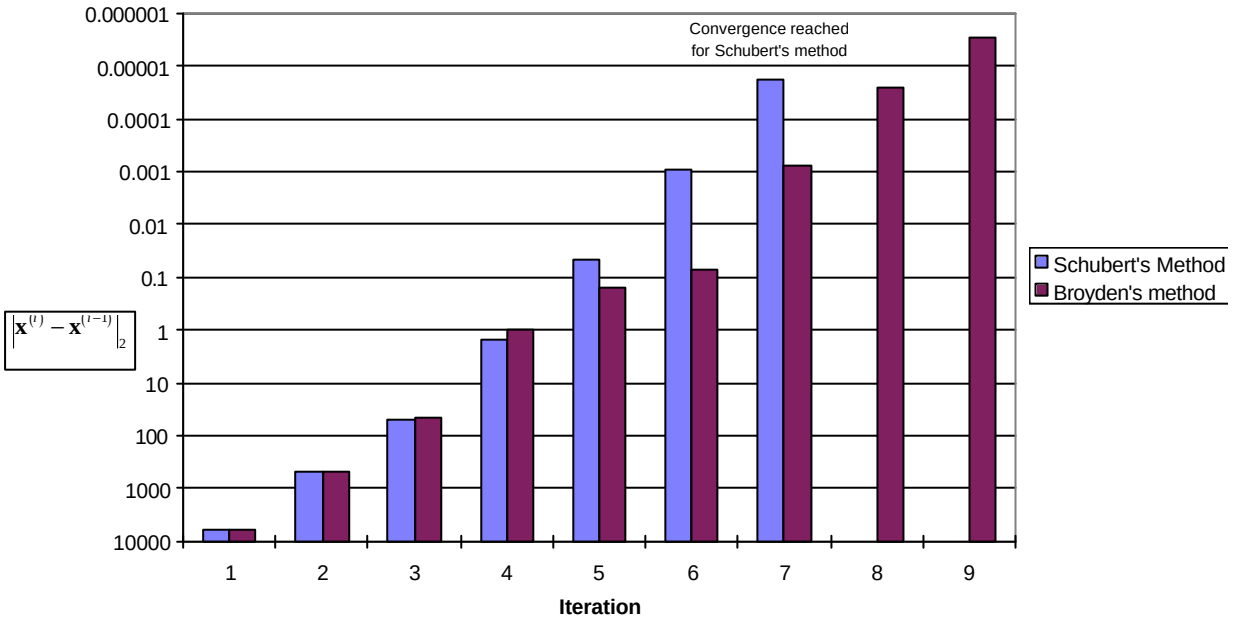


Figure 34: Convergence of Broyden's and Schubert's method for model KR4. Schubert's method converges in less iterations than Broyden's method.

To compare the 'quality' of the updates we do not want the method's to restart. Therefore we first compute a number of Newton iterations to compute a sufficient initial value for Schubert's or Broyden's to converge without a restart.
First we solve model KR4. The initial value is computed by 3 Newton iterations. The results are given in Figure 34. For the model KR8 the initial value is computed by 5 Newton steps. The results are presented in Figure 35.

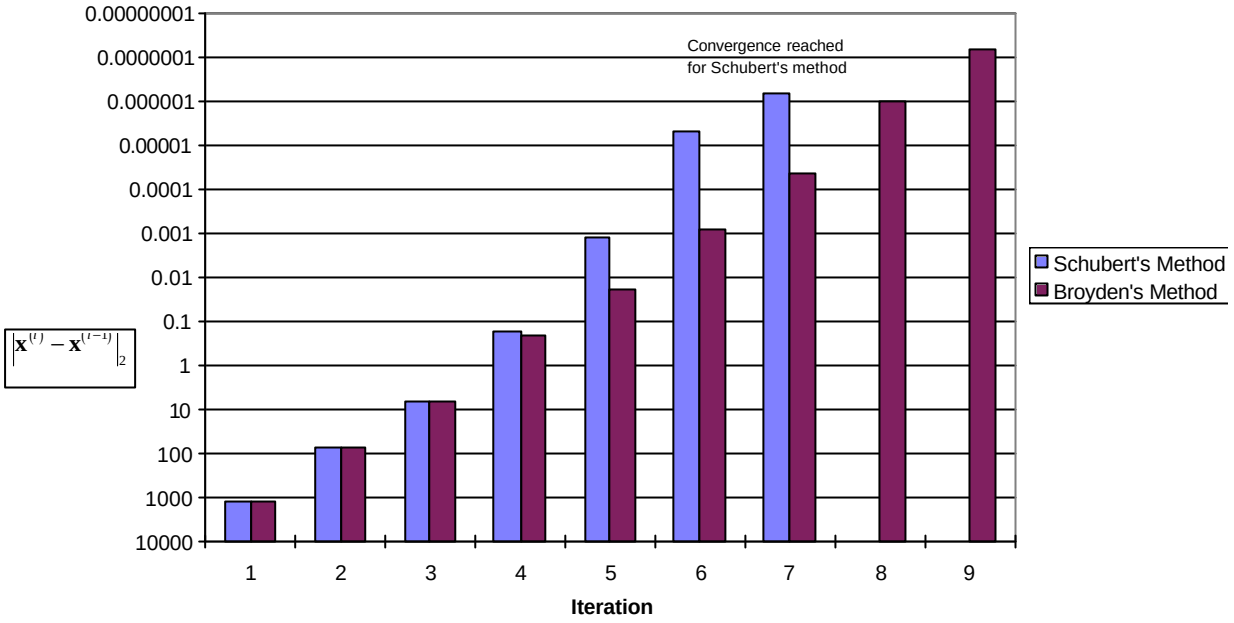


Figure 35: Convergence of Broyden's and Schubert's for model KR8. Schubert's method converges in less iterations than Broyden's method.

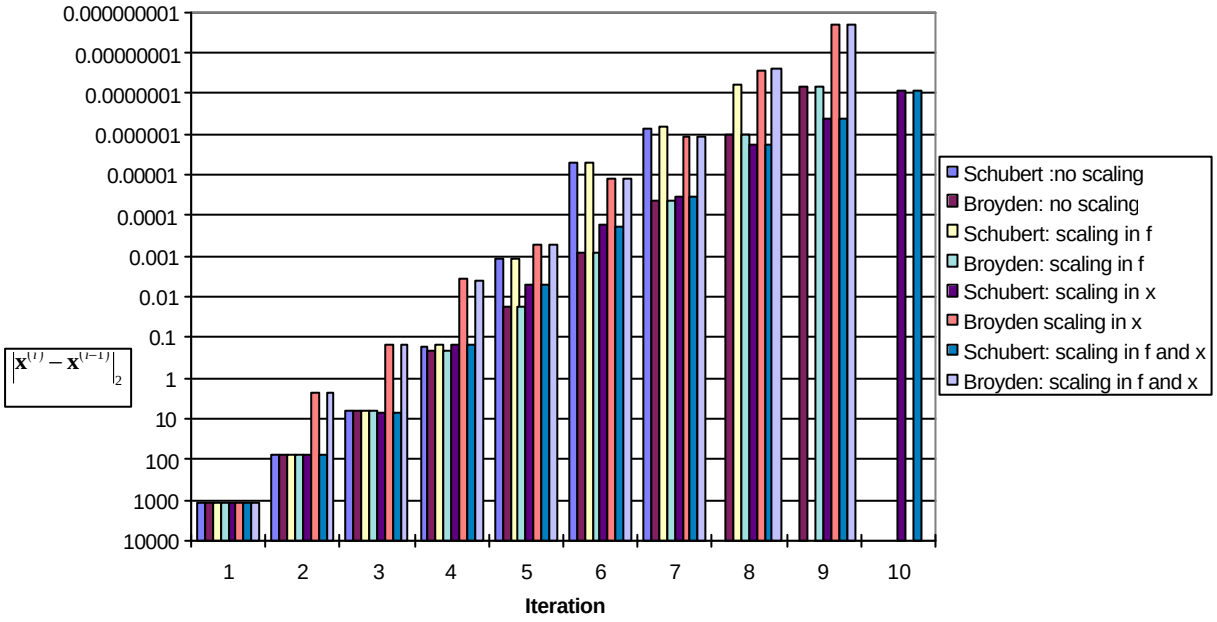


Figure 36 : The influence of scaling for Broyden's and Schubert's method for solving KR8.

For both problems, Schubert's method converges in less iterations than Broyden's method.
But Schubert's method needs to solve a system of linear equations at every step. Therefore in computation time Broyden's method is 2.3 times faster than Schubert's method for model KR4 and 2.1 times faster for model KR8.
Broyden's method and Schubert's method are both not scale invariant in **x**. Scaling is used for solving model KR8. The results are given in Figure 36. If scaling in f is applied only a small difference occurs between the correction steps. This indicates that the path to the solution hardly changes by applying this type of scaling both for Schubert's as well as for Broyden's method. For Broyden's method this is

proven in §4.7. Scaling in **x** however, results in different step-sizes and the number of iterations. In Figure 37 the results of scaling for solving model KR4 are presented. For Broyden's method scaling results in a faster convergence. Scaling in **x** for Schubert's method results for both tested models in a slower convergence than without scaling.

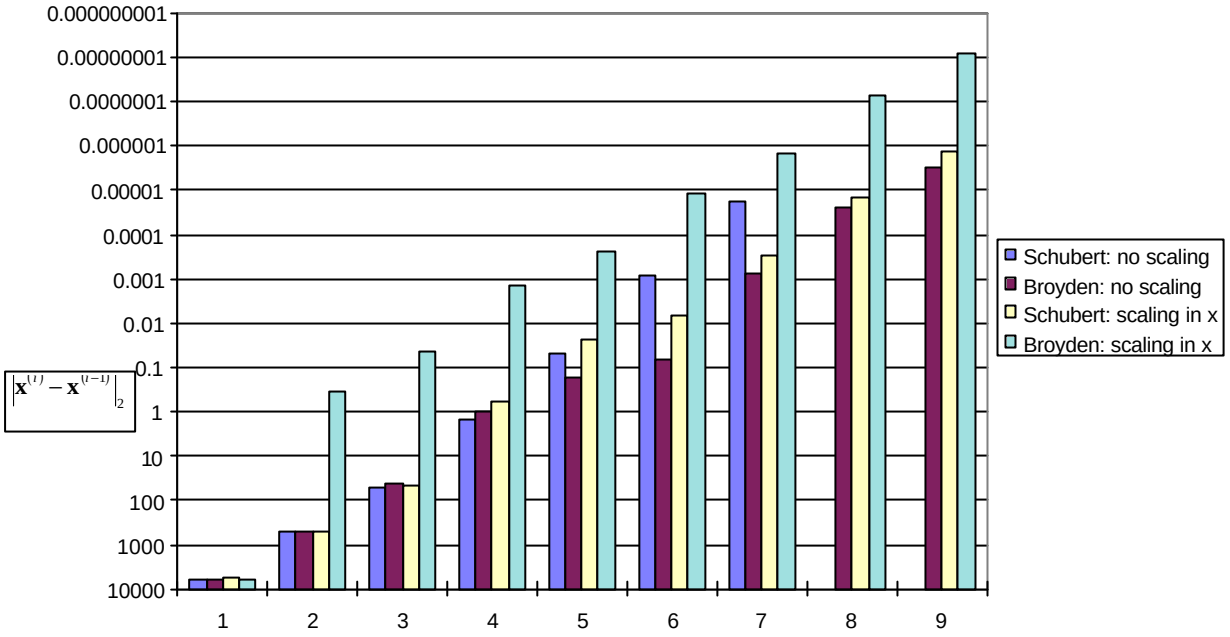


Figure 37:The influence of scaling for Broyden's and Schubert's method for KR4.

The updates of Schubert's method without scaling are better than the updates of Broyden's method. Unfortunately Schubert's method has to solve a system of linear equations at every iteration and therefore needs more computation time than Broyden's method. Schubert's method will probably be very useful for sparse systems of nonlinear equations of which the LU-decomposition cannot be computed or stored due to a huge amount of fill-in. For these types of problems the linear system is solved iterative and Schubert's update can be used instead of Broyden's update. For the type of problems we have tested Broyden's method can be very useful. Newton's method versus Broyden's method in computational time will be tested in §6.7 for a model of the same size as the Open SPYRO problem.

6.5 Stiff Models

The Froment model does not describe the radical reactions and is not very stiff. To test the solver for stiff models, a parameter a is introduced for every BLOCK of the PER model. With this parameter all reactions can be set a -times faster in each BLOCK. By increasing a the stiffness of the PER model increases. Some of the solutions of one BLOCK of the PER(a) model are given in Figure 38.

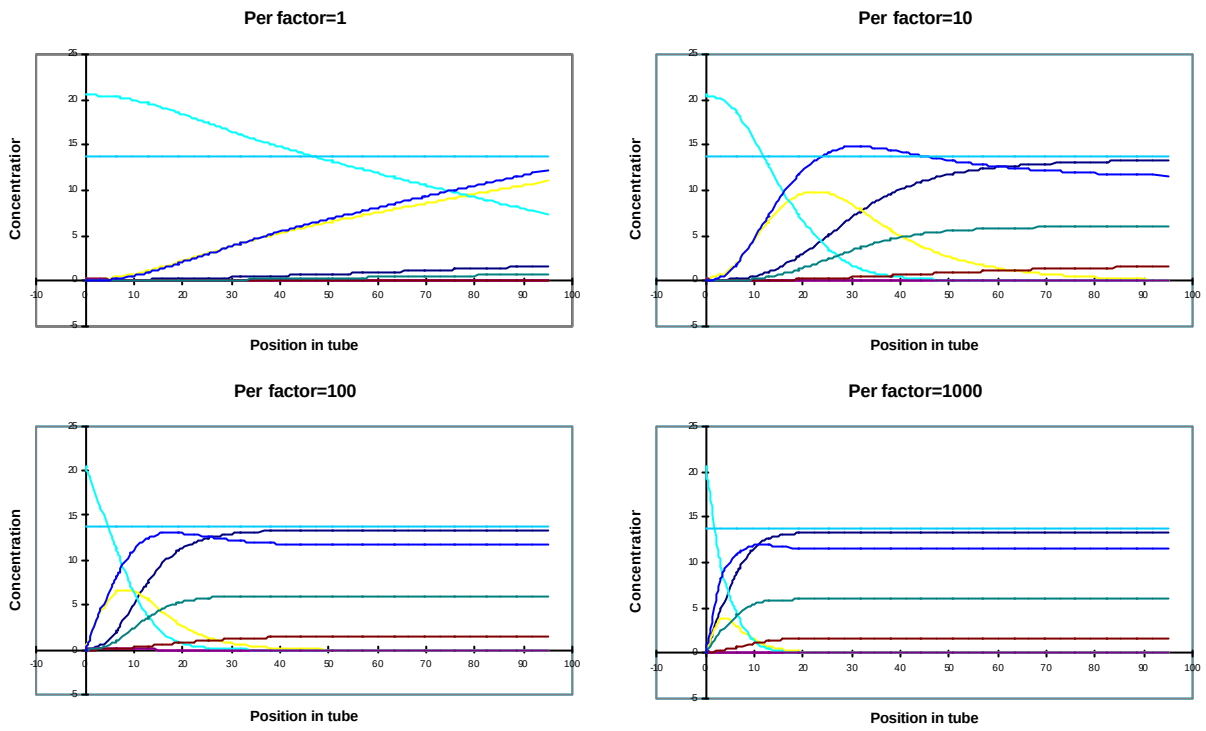


Figure 38: The solutions of some of the adjusted models per.

The methods for solving stiff nonlinear equations are tested on the model PER with 15 sections and 3 collocation points. The dimension of this system is 5060.
If $a > 2$ Newton's, Broyden's and Schubert's method do not converge. Euler integration is used to calculate an appropriate initial value for Newton's method to converge. The number of Euler steps is increased with 5 steps, whenever Newton's fails to converge. The same strategy is used to increase the number of Newton homotopy steps whenever necessary. The results of Euler integration are given in Figure 39.
The second pre-solver, Newton homotopy is tested similarly. To avoid unnecessary iterations per homotopy step we must set a stopping criterion per homotopy step. The stopping criterion $\left|\mathrm{h}\left(\mathbf{x}, t_{step}\right)\right|_2 < 1.0$ has been proven to be a good choice for the tested problem.
The results of Newton homotopy are given in Figure 40.
Note that the computation time of one Euler integration step of one Newton homotopy step equals the time to compute one Newton step.

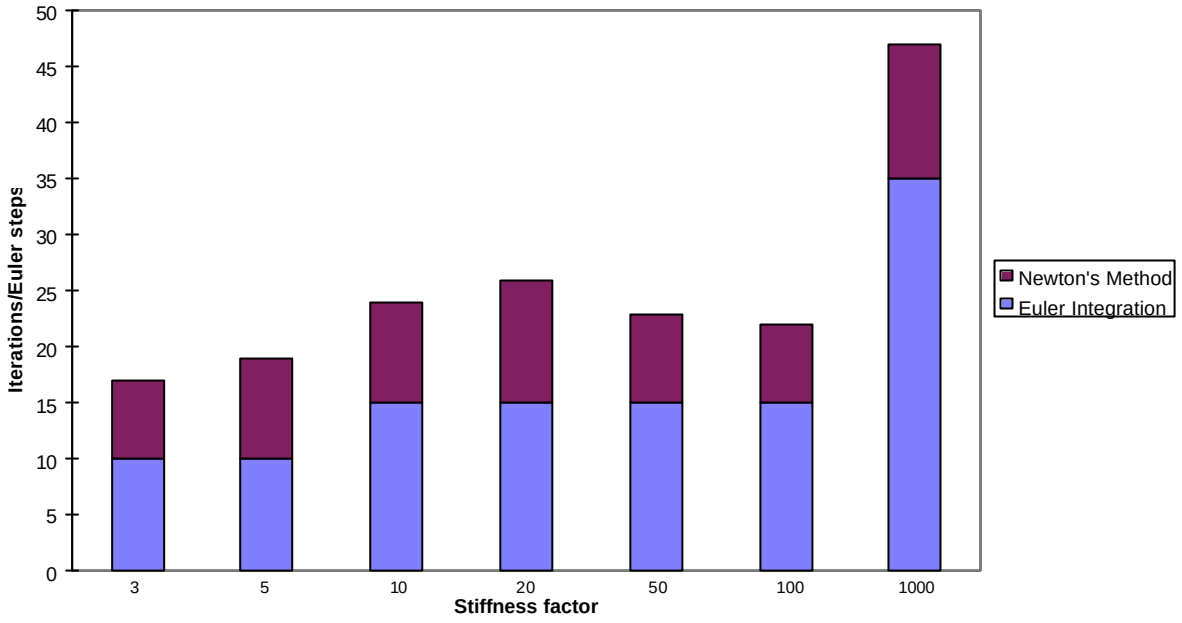


Figure 39: Solving the stiff version of model PER using Euler integration to compute a sufficient initial guess for Newton's method to converge. The computational time of one Newton step equals the time to compute one Euler step. Therefore we can add the number of Newton steps and Euler steps.

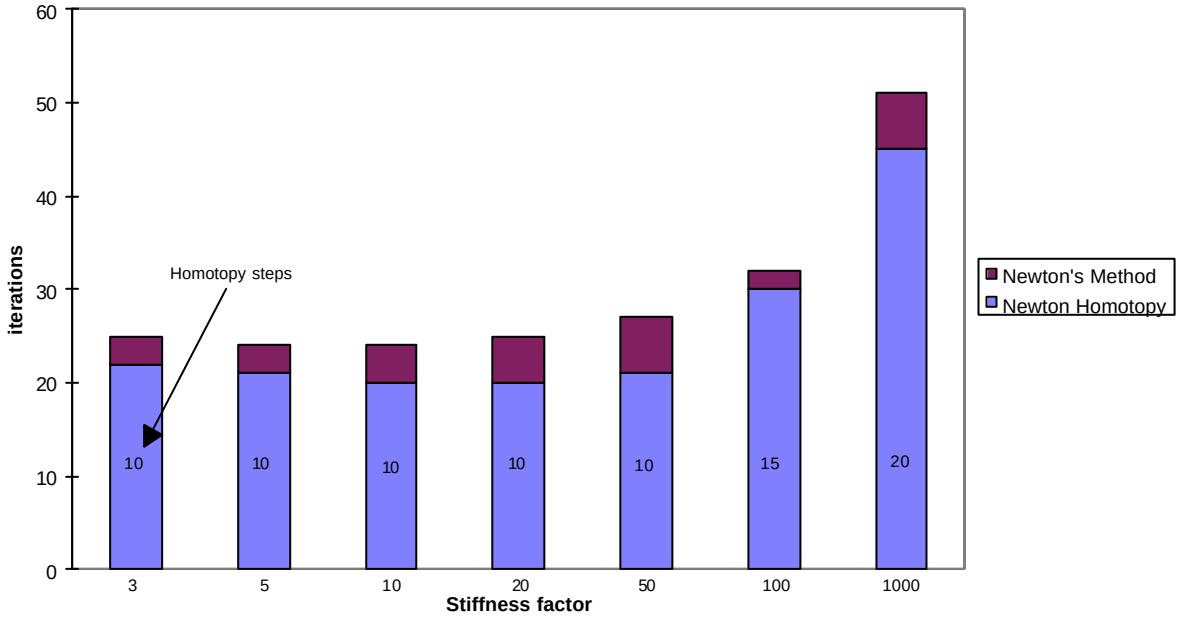


Figure 40:Solving the stiff version of model PER using Newton homotopy to compute a sufficient initial guess for Newton's method to converge. The computational time of one Newton iteration equals the time to compute one Newton homotopy step. Therefore we can add the number of Newton iterations and Newton homotopy steps.

If we try to solve model PER with a stiffness factor of 3.0 using Newton's method with domain damping, we need over 50 iterations. Hence this method is not useful for the tested problem. A much faster convergence is achieved if domain damping with reinitialization is used. The domain of reinitialization D is defined by: $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in D$ when $x_i \geq 0$ for all i. The domain of damping $D + D_e$ is chosen as $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in D + D_e$ when $x_i \geq -10^{-1}$ for all i. The results are given in Figure 41.
These choices proved to be a good choice for the PER model.

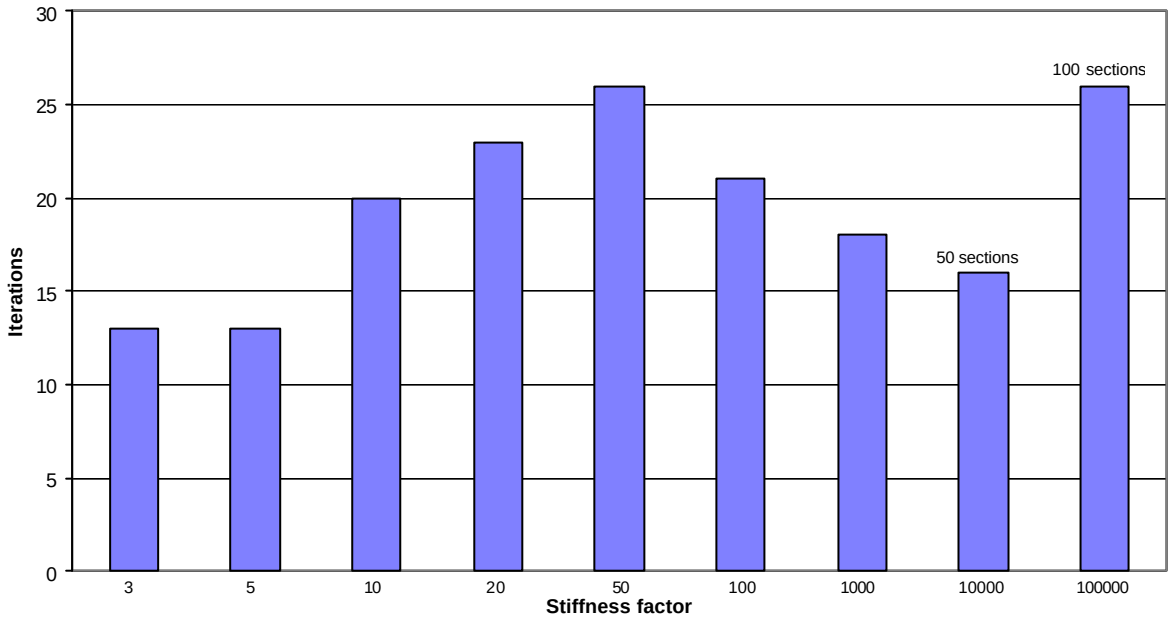


Figure 41: Solving the stiff version of model PER using Newton's method with domain damping and reinitialization. For a stiffness factor exceeding 1000 we need more sections to approximate the solution sufficiently.

For $a > 1000$, more sections are needed to describe the model accurately.

For stiff problems we can use Euler integration, Newton homotopy and domain damping with reinitialization. The fastest method is Newton's method with reinitialization but for this method we need a domain of reinitialization and a domain of damping before we can solve the system of nonlinear equations.
Therefore we will use Newton's method with domain damping when information about the domains is available. If this information is not available we can use Newton homotopy or Euler integration. The advantage of Euler integration is that it converges faster and we only have to choose the number of steps and not a stopping criterion per homotopy step.

6.6 Scaled Norms

Scaled norms can be a useful tool to stop the iteration procedure at the correct time. In this paragraph we will discuss the use of scaled norms as described in § 5.4.1. The differences between the scaled and non-scaled norms were compared for the last iterations before convergence was reached. The results are illustrated in Figure 42.

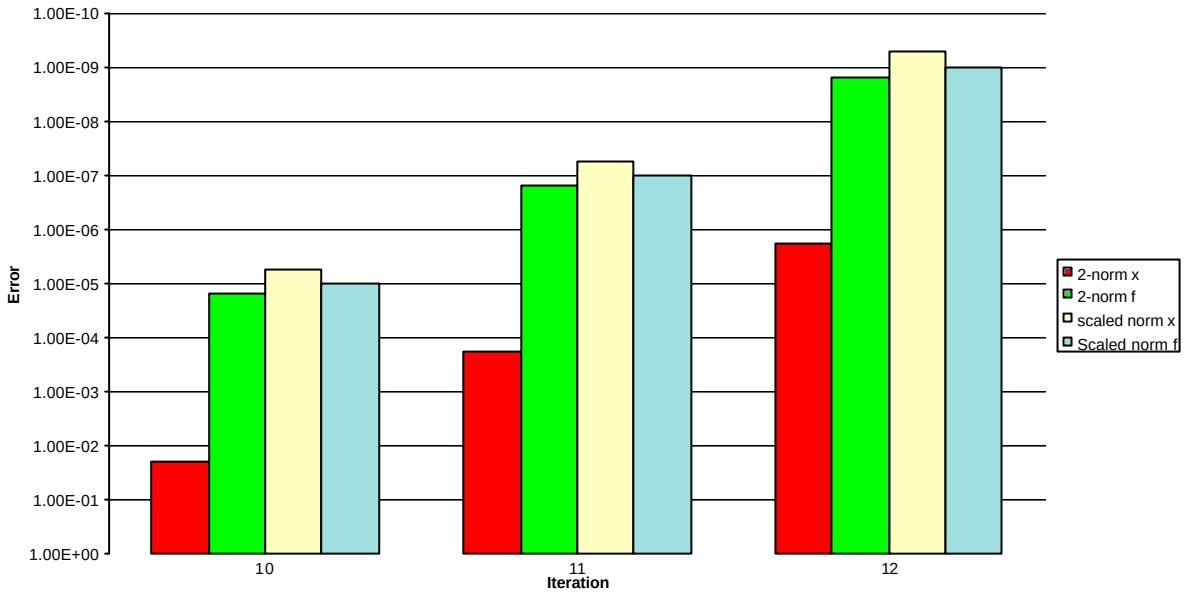


Figure 42: Last iterations of Newton's method using different types of norms for solving model KR1.

Close to the solution Newton's, Broyden's and Schubert's method converge at least super-linear. Therefore we can use the order of the last computed correction in **x** as a stopping criteria. If we want k significant figures for every component of **x** we can set the stopping criterion 10^{-k} for the scaled norm in **x** as defined in §5.4.1.
For the type of scaled norm in **x** that we use NOWAK(1991) suggested to choose the stopping criterion $10^{-k}\sqrt{\dim}$ if we want k significant figures for every component. This stopping criterion is tested and proved to be sufficient for the steam cracking models. Note: in most cases the stopping criterion $10^{-k}\sqrt{\dim}$ resulted in the same number of iterations as the stopping criterion 10^{-k}. For the steam cracking models we see that close to the solution the scaled norm in **x** is always larger than the scaled norm in f (see Figure 42[2]).
For the steam cracking models, a sufficient stopping criterion is to stop iterating whenever (i) the scaled norm in f is smaller than $10^{-k-1}\sqrt{\dim}$ and (ii) the scaled norm in **x** is smaller than $10^{-k}\sqrt{\dim}$.

[2] The results of the other models are similar. Therefore we only present the results of scaling norms for the KR1 model

6.7 An Approximation of the Computational Time for the Open SPYRO Model

The Open SPYRO model consists of 124 components. The Froment model consists of only 9 components. The Froment model is adjusted by adding a number of extra equations. In this way a model is constructed consisting of 126 components (this is a multiple of 9). In every section the solution is approximated by a polynomial of degree at most three. We will compare Newton's method and Broyden's method for this model. We measured the computation time for a variety of sections because up to now it is not known at this time how many sections are needed for the Open SPYRO model.

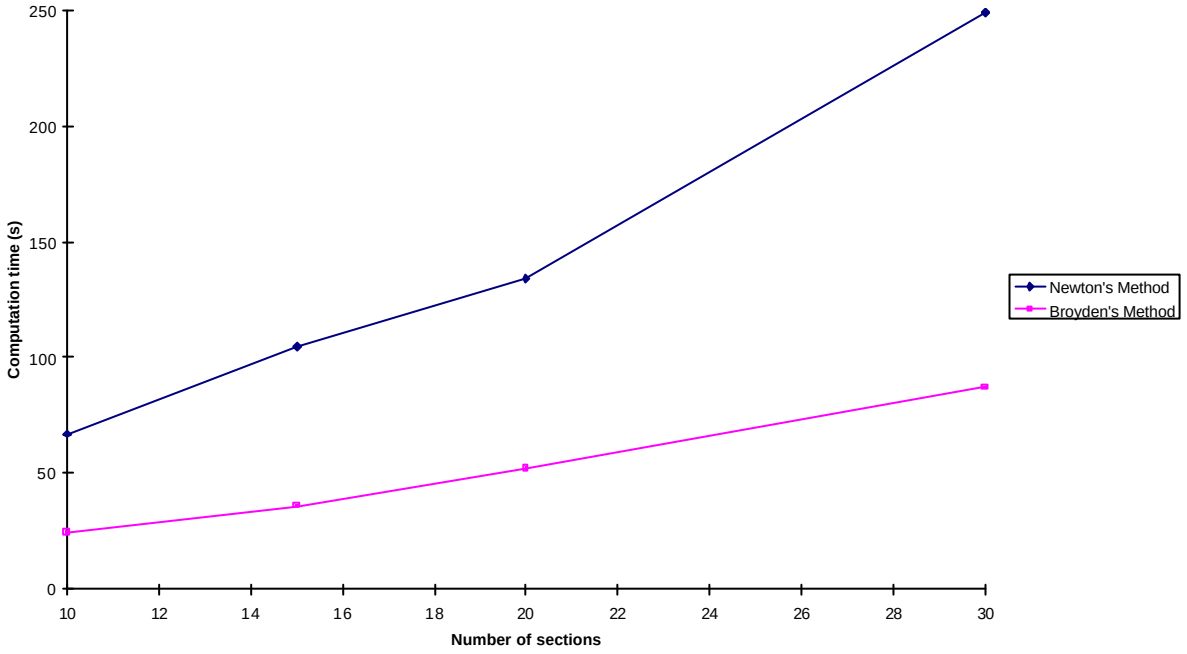


Figure 43: Time to compute a model of the size of SPYRO. Horizontal profiles are used as an initial guess.

For the computation we used a PC with a PENTIUM II processor at a speed of 266MHz and 64Mb internal memory . The results are presented in Figure 43. We see that Broyden's method converges faster than Newton's method. The number of iterations for both methods are equal for all four test problems. Broyden's method computed 11 updates, restarts, and converges after computing another 14 updates. Newton's method converges in 8 iterations.
An approximation of the computation time per iteration for Broyden's method and Newton's method are given in Figure 44 and Figure 45. Note that before computing a number of Broyden's updates we first have to compute one Newton step.

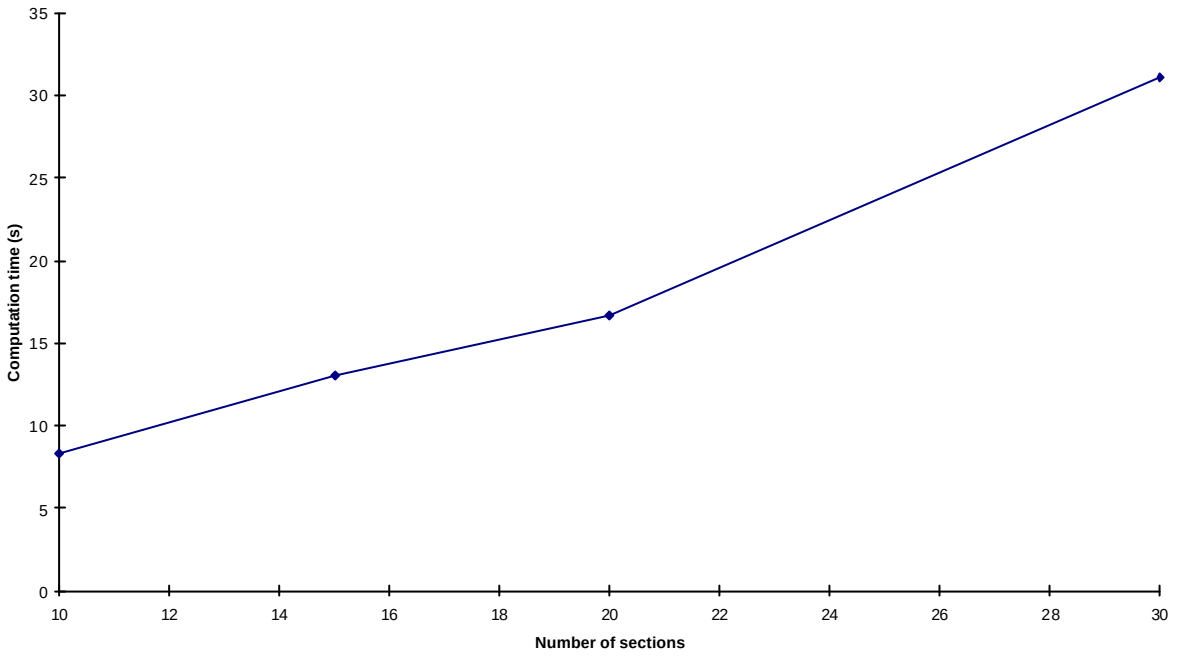


Figure 44:An approximation of the computational time of one Iteration of Newton's method

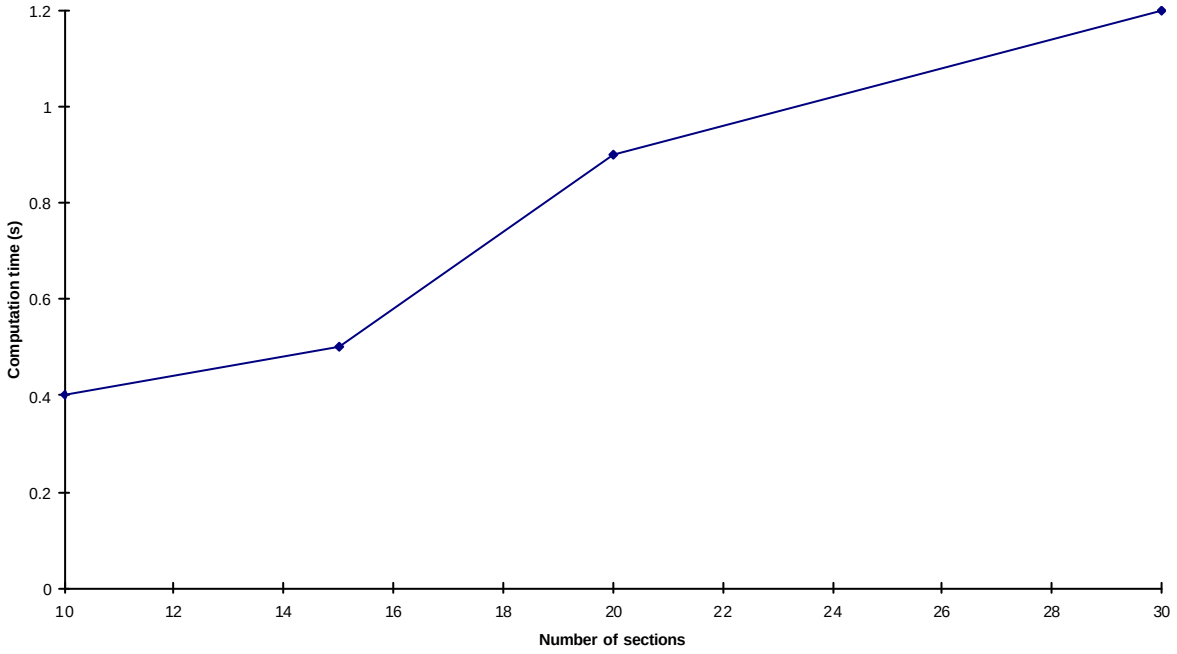


Figure 45:An approximation of the computational time of one Iteration of Broyden's method

For all the test we have used a bad initial guess. For the open SPYRO problem better initial guesses will be used. A user of SPYRO is interested in the influence on the cracking process when different settings of the process, like the feed, temperature, pressure, etc. are changed. This justifies the following test for an indication of the computation time for solving the Open SPYRO problem. The problem PR2 (with 20 sections) was solved. The computed solution is stored and will be used as an initial guess for solving the PR2 model with different settings. We have changed the feed, the

temperature, and the pressure up to 20%. If the solution of the original problem was used as an initial guess we achieved a much faster convergence. Broyden's method did not restart after a number of iterations and converged in a average of 25 seconds.

7 CONCLUSIONS AND RECOMMENDATIONS

The conclusions and recommendations have been divided into three sections. The first section deals with the solver for sparse systems of linear equations. The second section is covering the solver for sparse systems of nonlinear equations. In the final section some recommendations for further research will be given.

7.1 The Solvers for Sparse Linear Systems of Equations

For the type of problems we have tested, the ILU-GMRES method is not suited. The ILU-GMRES solver only converges when the preconditioning is very good. Therefore the time needed to compute the pre-conditioned almost equals the time to solve the system of linear equations directly. The three direct methods, Householder's method, the KTI solver and the LU-decomposition produces accurate solutions to the tested problems. Householder's method and the KTI solver are much slower than the LU-decomposition algorithm. I suspect that it is not the differences in number of computations, but the use of memory that is responsible for the differences in computation time between the different methods.
Householder's method is slower than the KTI solver but produces better results. In all aspects the LU-decomposition performs better. Therefore it can be concluded that the LU-decomposition solver is to be used for slving sparse systems of nonlinear equations.

7.2 The Solvers for Sparse Systems of Nonlinear Equations

We have tested Newton's, Broyden's and Schubert's method, Euler integration and Newton homotopy for solving systems of nonlinear equations. Newton's method needs the smallest number of iterations. At every step, the Jacobian matrix must be computed and a linear system of equations has to be solved. For all the non-stiff models we have tested, Newton's method converges if damping techniques are used. In general, Newton's method with standard Deuflhard damping needs less iterations and converges in less time than Newton's method using other damping techniques. Therefore this damping technique is preferred for Newton's method. Broyden's method needs the largest number of iterations. This method updates the Jacobian matrix by using a recursive algorithm. Broyden's method can be adjusted for large sparse Jacobian matrices. The advantage of this method is that we can use the previous Jacobian matrix and its LU-decomposition for a number of iterations. Therefore Broyden's method converges in less time to a solution than Newton's method. Schubert's method updates the Jacobian matrix by using the sparsity structure of the previous Jacobian. If scaling is not used Schubert's update results in a faster convergence than Broyden's method. Unfortunately, we cannot use the LU-decomposition of a previously computed iteration to compute the system of equations as in Broyden's method. Therefore a linear system equations must be solved at every step. The computation time gained by the better updates of Schubert's method does not compensate the additional computations to solve the systems of linear equations at every step. Schubert's method is the slowest method of all the tested methods.
For very stiff problems we need a good initial guess of the solution. This initial guess can be computed by using Euler integration or Newton homotopy. A drawback of these methods is the huge amount of computations needed to compute a sufficient accurate initial guess.
Newton's method with domain damping and reinitialization is very useful for the stiff systems tested. This method needs less computations than the continuation methods. A drawback of this method is that we need a domain of the variables.

To stop the iteration process of the iterative methods for solving systems of nonlinear equations at the right time, we use scaled norms and combined stopping criteria.
Scaling does not effect the results of Newton's method. For the tested problems, Broyden's method with scaling converges faster than without scaling.
Of the non-stiff problems we have tested Broyden's method performs best. For stiff problems Newton's method with domain damping and reinitialization can be used.

All the tested problems could be solved. A non-stiff problem with the size of Open SPYRO and a maximum of 30 sections, this is a nonlinear system of equations with a dimension of 11708, can be solved within two minutes on a 266MHz Pentium II PC by using Broyden's method. If the Open SPYRO problem is too stiff to solve directly by using Newton's Method or Broyden's method, there are tools available to produce an appropriate initial value.

It can be expected that the Open SPYRO problem with 20 sections and polynomials of at most degree two can be solved within 25 seconds, when better initial guesses are constructed.

7.3 Recommendations

The initial value for all the tested models are horizontal profiles equal to the input of every component. The computational time can be reduced if better initial values are produced. This has been shown for the PR2 model in §6.7.
Domain damping and reinitialization is at this time only possible for the positive domain.
A generic solver must be equipped with a more advanced option for the definition of the domain of the variables. The speed of the implementation is likely to be improved if the code is optimized.

8 REFERENCES

1. F. L. Bauer. Optimally scaled matrices. *Numer. Math.* 5:73-87, 1963.
2. C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Math. Comp.* 19:577-593, 1965.
3. H. N. Cofer and M. A. Stadtherr. Reliability of iterative linear equation solvers in chemical process simulation. *Comp. Chem. Eng.* 20(9):1123-1132, 1996.
4. T. H. Corman, C. E. Leiserson and R. L. Rivest. Introduction to algorithms. *Mc Graw Hill* 18th print ISBN 0-07-013143-0, 1997.
5. J. E. Dennis Jr. and J. J. Moré. Quasi-Newton methods motivation and theory. *Siam Rev.* 19:47-89, 1977.
6. P. Deuflhard. A modified Newton method for the solution of ill-conditioned systems of nonlinear equations with applications to multiple shooting. *Numer. Math.* 22:289-315, 1974.
7. I. S. Duff, A. M. Erisman and J, K, Reid. Direct methods for sparse matrices. *Oxford Sc. Publ.*, 1986.
8. A. C. N. Duin. Parrallel sparse matrix computations. ISBN 90-9011435-1, 1998.
9. G. F. Froment and K. B. Bischoff. Chemical reaction analysis and disign. *John Wiley & Sons,Inc.*, ISBN 0-471-02447-3, 1979.
10. J.R. Gilbert, C. Molder and R. Schreiber. Sparse matrices in Matlab: Design and implementation. *Matlab 5.2 documentation*, 1991.
11. G. H. Golub and H. A. van der Vorst. Closer to the solution: Iterative linear solver. *in preparation*, 1997?
12. E. Marwil. Convergence results for Schubert's method for solving sparse nonlinear equations. *Siam J. Num. Anal.* 16(4):588-604, 1979.
13. U. Nowak and L. Weimann. A Family of Newton codes for systems of highly nonlinear equations. *Technical report* TR-91-10 *Konrad-Zuse Zentrum für Informationtechnik Berlin*, 1991.
14. J. R. Paloschi and J. D. Perkins. An implimentation of quasi-newton methods for solving sets of nonlinear equations. *Comput. Chem. Engng.*12(8):767-776, 1988.
15. Y. Saad. Sparskit II a basic tool kit for sparse matrix computations. 1994.
16. R. Tewarson. Sparse matrices. *Academic Press*, 1973.
17. J. Villadsen and M. L. Michelsen. Solution of differential equation models by polynomial approximation. *Printice-Hall, Inc.*, 1978.
18. H. A. van der Vorst and T. F. Chan. Linear system solver: Sparse iterative methods. *in preparation*, 1997?

APPENDIX A

FROBENIUS NORM

The Frobinius norm $|\mathbf{A}|_F$ of the matrix

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Is defined by:

$$|\mathbf{A}|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{i,j}^2}$$

SHERMAN-MORRISSON FORMULA

Theorem (Sherman-Morrisson formula) Let **u** and **v** are column vectors in $\mathbf{R}^n$, and **A** is a non-singular $n \times n$ matrix. If

$1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u} \neq 0$ then $\mathbf{A} + \mathbf{u}\mathbf{v}^T$ is non-singular and

$$\left(\mathbf{A} + \mathbf{u}\mathbf{v}^T\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{u}\left(1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)^{-1}\mathbf{v}^T\mathbf{A}^{-1}$$

Proof:
The Sherman-Morrisson formula can be derived in the following way:

$$\begin{aligned} \mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T &= \mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\left(1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)\left(1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)^{-1}\mathbf{v}^T \\ &= \mathbf{I} + \left(\mathbf{A}^{-1}\mathbf{u} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)\left(1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)^{-1}\mathbf{v}^T \\ &= \mathbf{I} + \left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\right)\mathbf{A}^{-1}\mathbf{u}\left(1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)^{-1}\mathbf{v}^T \end{aligned}$$

and therefore

$$\mathbf{I} = \left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\right)\left(\mathbf{I} - \mathbf{A}^{-1}\mathbf{u}\left(1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)^{-1}\mathbf{v}^T\right)$$

we can conclude that $\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\right)$ is non-singular and

$$\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\right)^{-1} = \left(\mathbf{I} - \mathbf{A}^{-1}\mathbf{u}\left(1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}\right)^{-1}\mathbf{v}^T\right).$$

We can now derive the Sherman-Morrisson formula

$$\begin{aligned} \left(\mathbf{A} + \mathbf{u}\mathbf{v}^T\right)^{-1} &= \left(\mathbf{A}\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\right)\right)^{-1} \\ &= \left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\right)^{-1}\mathbf{A}^{-1} \\ &= \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{u}\left(\mathbf{I} + \mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\right)^{-1}\mathbf{v}^T\mathbf{A}^{-1}. \end{aligned}$$

(P,Q) BAND-MATRIX

Definition *A band-matrix of the type (p,q) is defined by*

$$A = \begin{pmatrix} a_{1,1} & \cdots & \cdots & a_{1,(q+1)} & 0 & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & \vdots \\ a_{(p+1),1} & & & & & \ddots & 0 \\ 0 & \ddots & & & & & a_{(n-q),n} \\ \vdots & \ddots & \ddots & & & & \vdots \\ \vdots & & \ddots & \ddots & & & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n,(n-p)} & \cdots & a_{n,n} \end{pmatrix}$$

APPENDIX B

The Counting sort algorithm

Counting sort assumes that each of the n input elements is an integer in the range 1 to k, for some integer k. When k=$O(n)$, the sort runs in $O(n)$ time.
The basic idea of counting sort is to determine, for each input element x, the number of elements less than x. This information can be used to place element x directly into its position in the output array. For example, if there are 17 elements less than x, then x belongs in the position 18. This scheme must be modified slightly to handle the situation in which several elements have the same value, since we don't want to put them all in the same position.
In the code for counting sort, we assume that the input array is an array $A[1..n]$, and thus $lengt[A]=n$. We require two other arrays: the array $B[1..n]$ holds the sorted output, and the array $C[1..n]$ provides temporary working storage.
The algorithm of counting sort is given in Figure 46.

```
For i=1 to k
        C[i]=0
Endfor
For j=1 to length[A]
        C[A[j]]=C[A[j]]+1
Endfor
For i=2 to k
        C[i]=C[i]+C[i-1]
Endfor
For j=length[A] downto 1
        B[C[A[j]]]=A[j]
        C[A[j]]=C[A[j]]-1
Endfor
```

Figure 46: The algorithm of counting sort

A thorough description of this method can be found in CORMEN(1997)

APPENDIX C

We will discuss the differences between the different test models:
KR1: is directly derived from the Froment model as described in FROMENT(1977). The friction is spread over the whole length of the coil. The other cracker models are based on this model. The models KR2-KR10 are variations of model KR1. These differences are briefly described below.

KR2: A different equation describing the pressure.
KR3: Bends in the tubes are modeled. There are two equations to describe the pressure profile (one for the bends and one for the straight part of the coil).
KR4: KR4 uses a more elegant momentum balance.
KR5: Different energy balance, no kinetic term.
KR6: Energy balance KR5, no kinetic term, different computation heat of reaction.
KR7: Different energy balance, no kinetic term.
KR8: Energy balance KR7, no kinetic term, different computation heat of reaction.
KR9: Different mass balances resulting in a sparser Jacobian matrix.
KR10: Same as KR8 different computation of enthalpy.

There are a number of large problems constructed to simulate the Open SPYRO problem.

PER: The equations of KR1 called BLOCKS are copied a number of times. In this way more components can be created. The different BLOCKS do not react with each other. For every BLOCK the speed of the reactions can be set to construct stiff systems.
PR2: The same as PER only the components in each BLOCK react with each other (This model is used to simulate SPYRO).
PR3: The same as PR2 but different mass balances are used resulting in a sparser Jacobian matrix.

Condition numbers: These are the Jacobian matrices of the models at the initial guess (horizontal profiles). For the models KR1,KR2,KR4-KR10 6 sections and 6 collocation points are chosen. KR3 has 13 sections and 6 collocation points. PER and PR2 have 3 collocation points and 4 sections and PR3 has 8 collocation points and 4 sections.

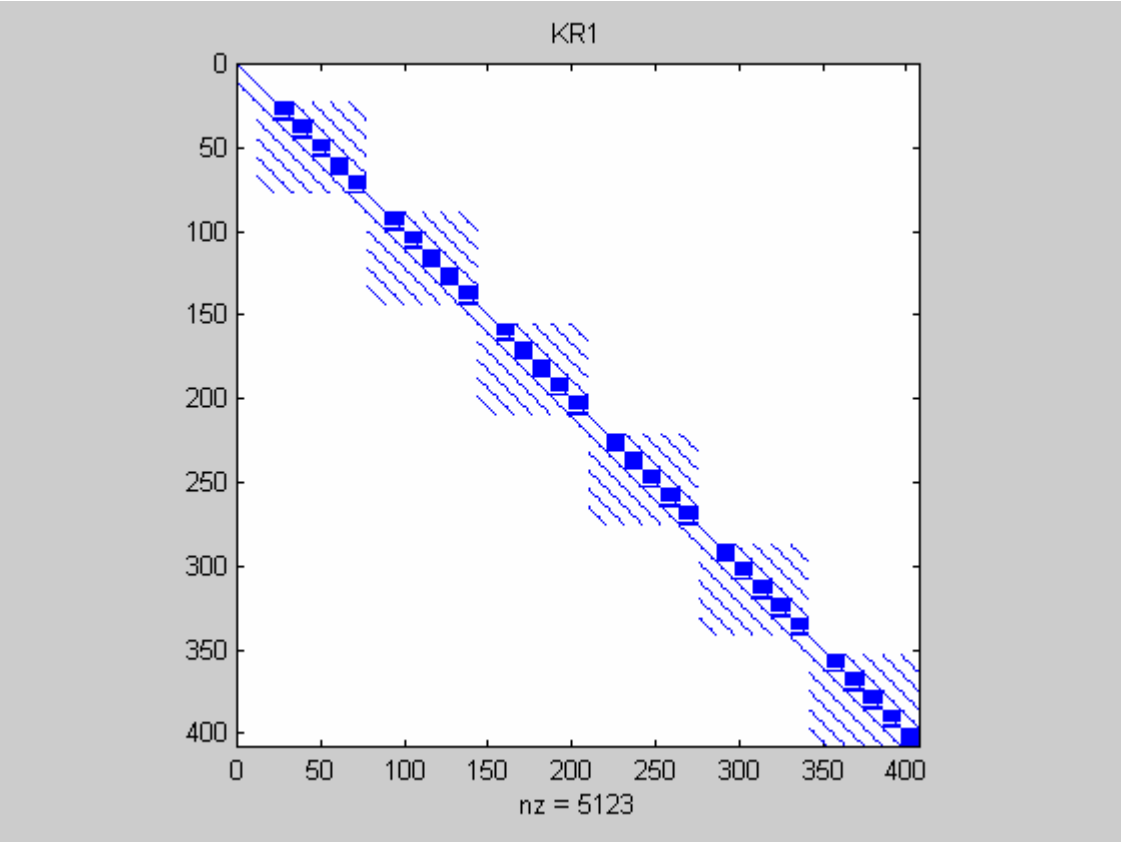
Condition numbers:

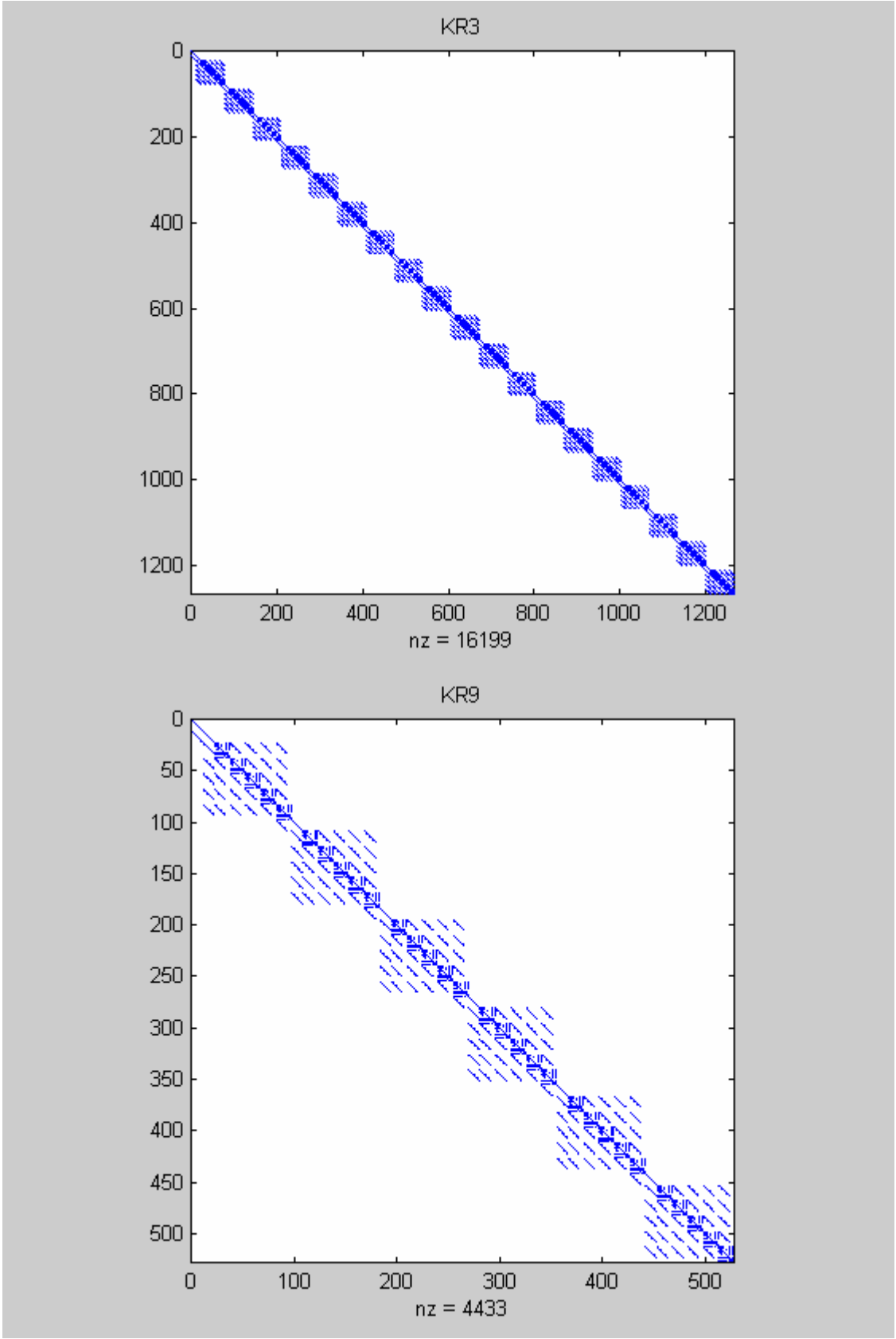
KR1: 1.1041E+8
KR2: 1.369693E+9
KR3: 9.7825E+8
KR4: 4.5645E+8
KR5: 2.5778E+12
KR6: 2.5958E+12
KR7: 4.0039E+11
KR8: 3.1136E+11
KR9: 2.8087E+11
KR10: 3.0366E+11
PR2: 7.2394E+11

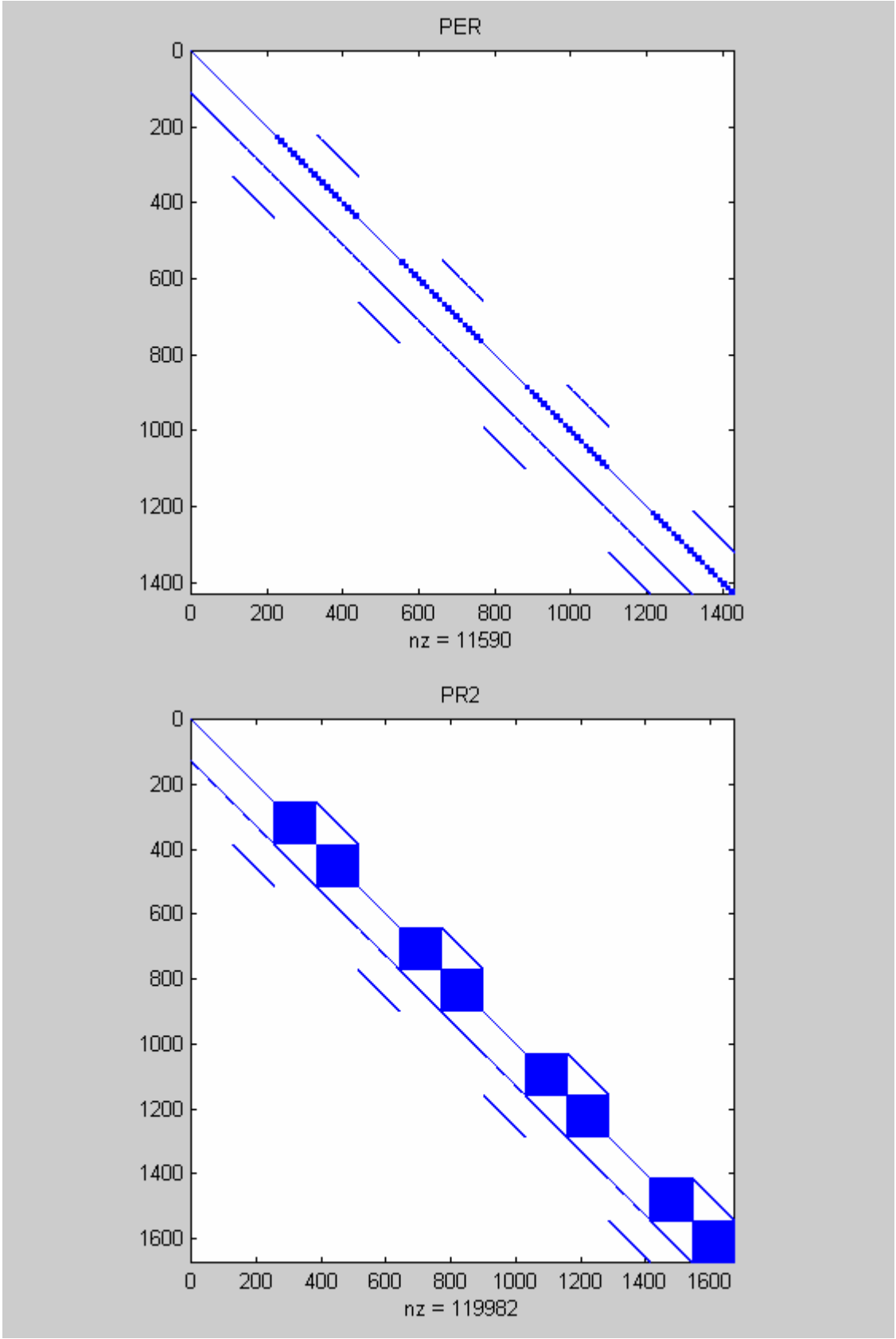
The condition numbers of PER and PR3 are not computed.
To compare the condition numbers, a number of random non-sparse matrices of the same size as KR1 are generated. The condition number of these matrices is in the order $O(10^4)$.
As a second comparison, random sparse matrices are generated. To ensure non-singularity a diagonal matrix with random elements is added to this sparse matrix. The order of the condition number of the

constructed matrices varies between $O(10^3)$ and $O(10^5)$. If we generate random matrices of the same sparsety structure as KR1 we get condition numbers between the order of $O(10^6)$ and $O(10^{10})$. From these tests we can conclude that the condition number does not primarily depend on the size of the elements but on the structure of the matrix.

The structure of the Jacobian matrices of the models KR1-KR10 are similar with the exception of KR3 and KR9.
The sparsity structures of the Jacobian matrices of KR1,KR3,KR9,PER,PR2 and PR3 are given below.




KR3
0
200
400
600
800
1000
1200
nz = 16199
KR9
0
50
100
150
200
250
300
350
400
450
500
nz = 4433


PER
0
200
400
600
800
1000
1200
1400
nz = 11590
PR2
0
200
400
600
800
1000
1200
1400
1600
nz = 119982